ISAAC SHEFF

HETEROGENEOUS CONSENSUS

Hi, I’m Isaac.
I’m a Ph.D. candidate here at Cornell, and
I’m here to talk about Heterogeneous Consensus, which is a project I’ve been working on

CLICK

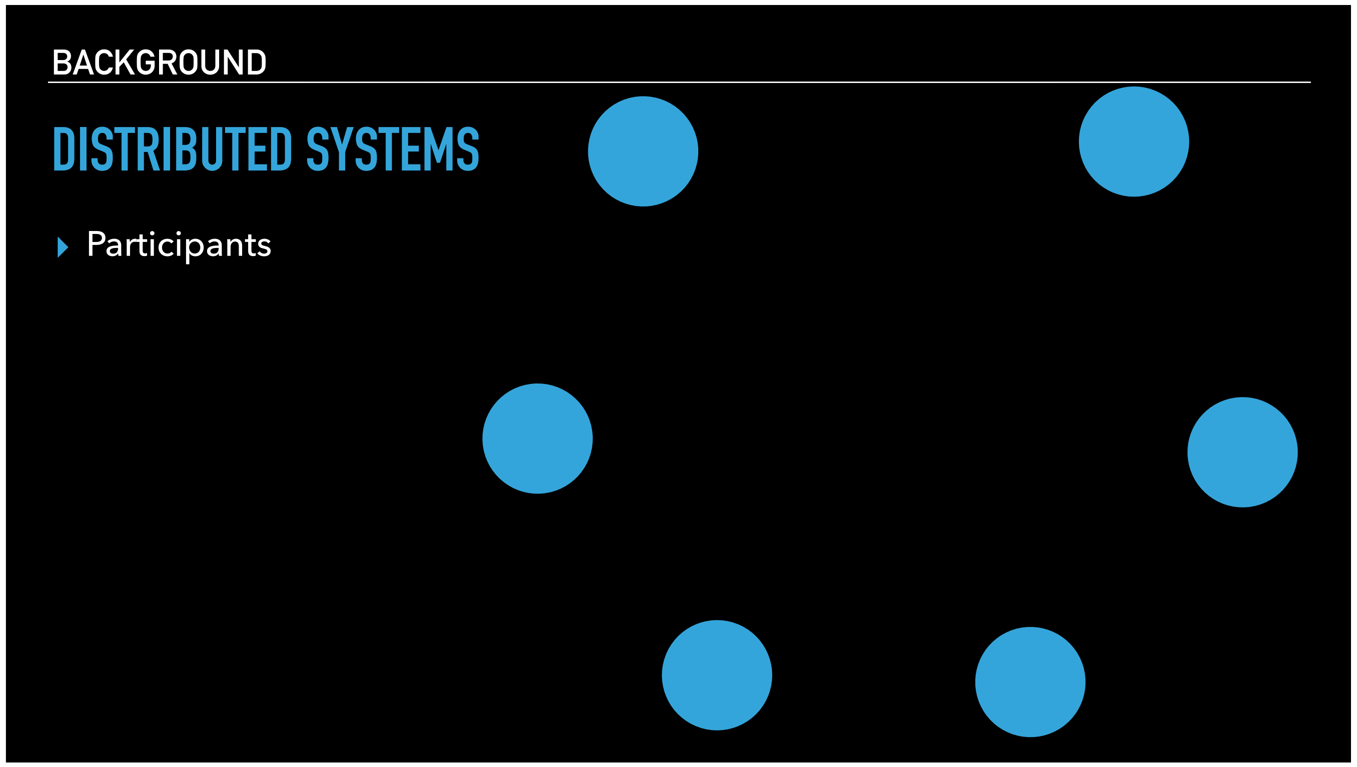
HETEROGENEOUS CONSENSUS

Isaac Sheff
isheff@cs.cornell.edu

Andrew C. Myers
andru@cs.cornell.edu

Robbert van Renesse
rvr@cs.cornell.edu

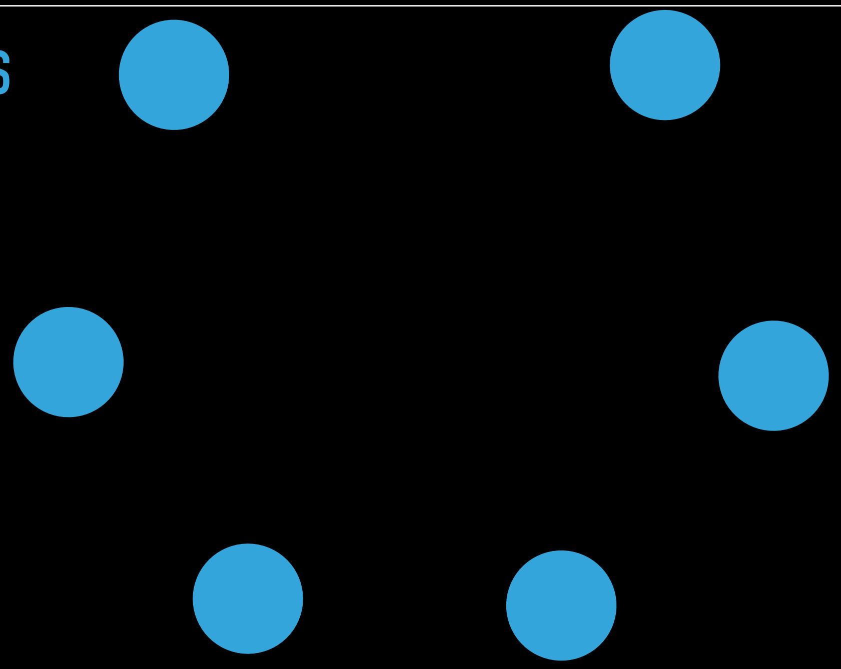
with my advisors Andru Myers and Robbert van Renesse



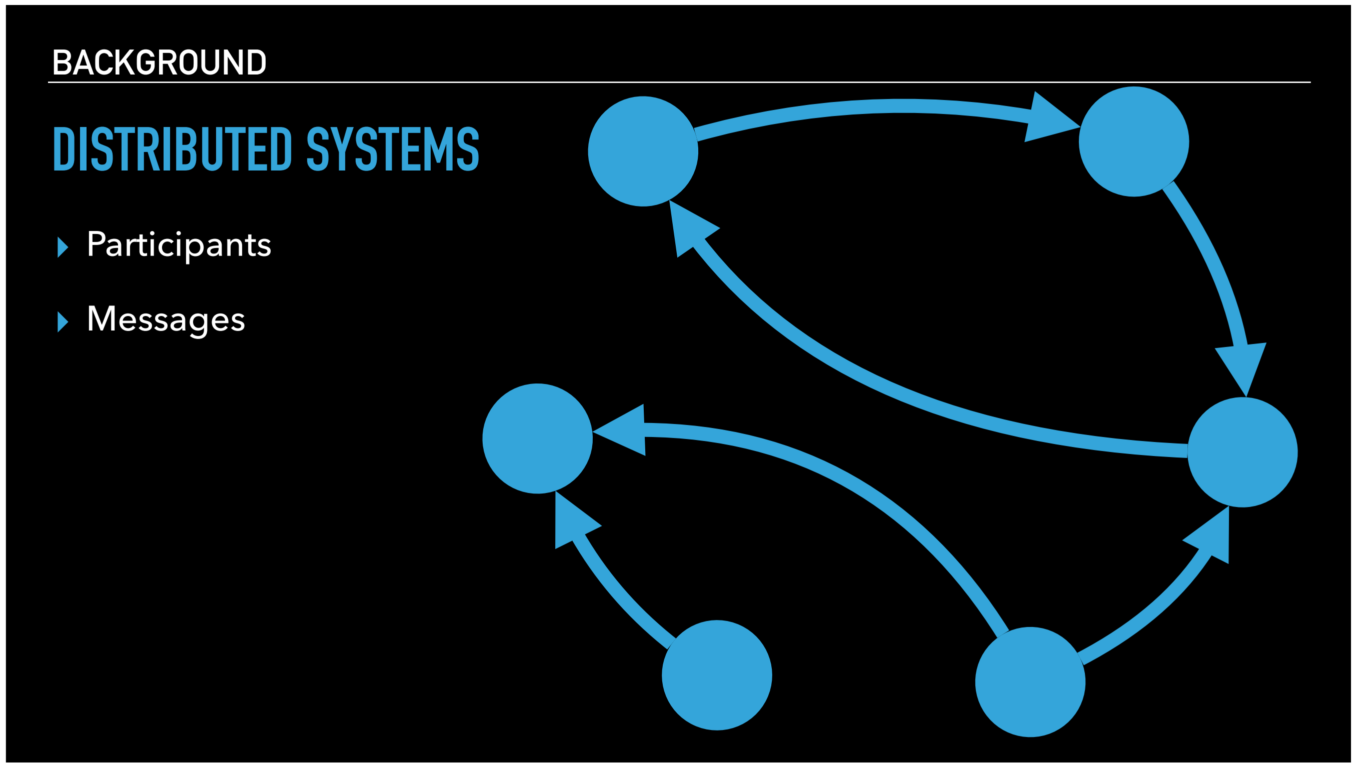
BACKGROUND

DISTRIBUTED SYSTEMS

- Participants



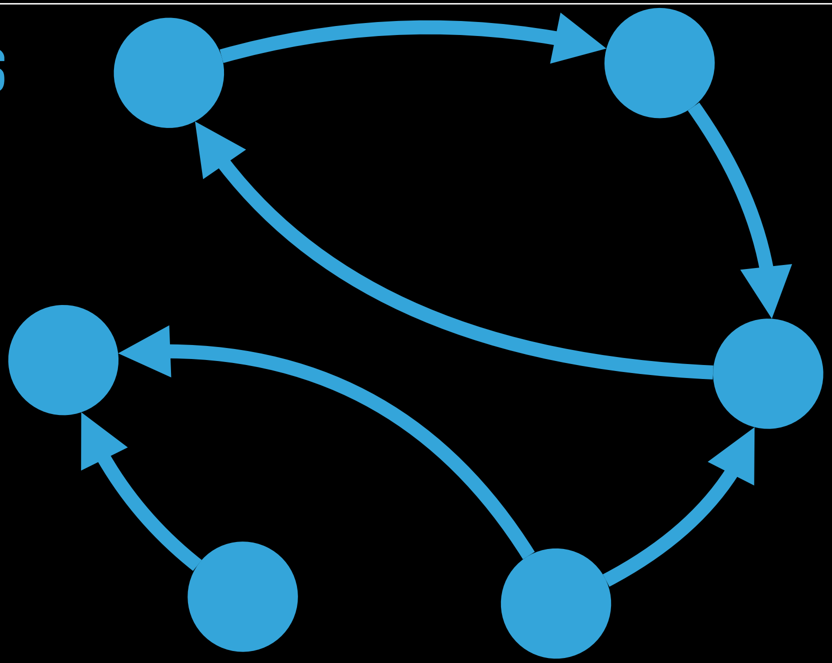
We work with Distributed Systems.
Classically, these are protocols and algorithms which assume there are a known set of participants
You can think of these as servers, nodes, people, that kind of thing



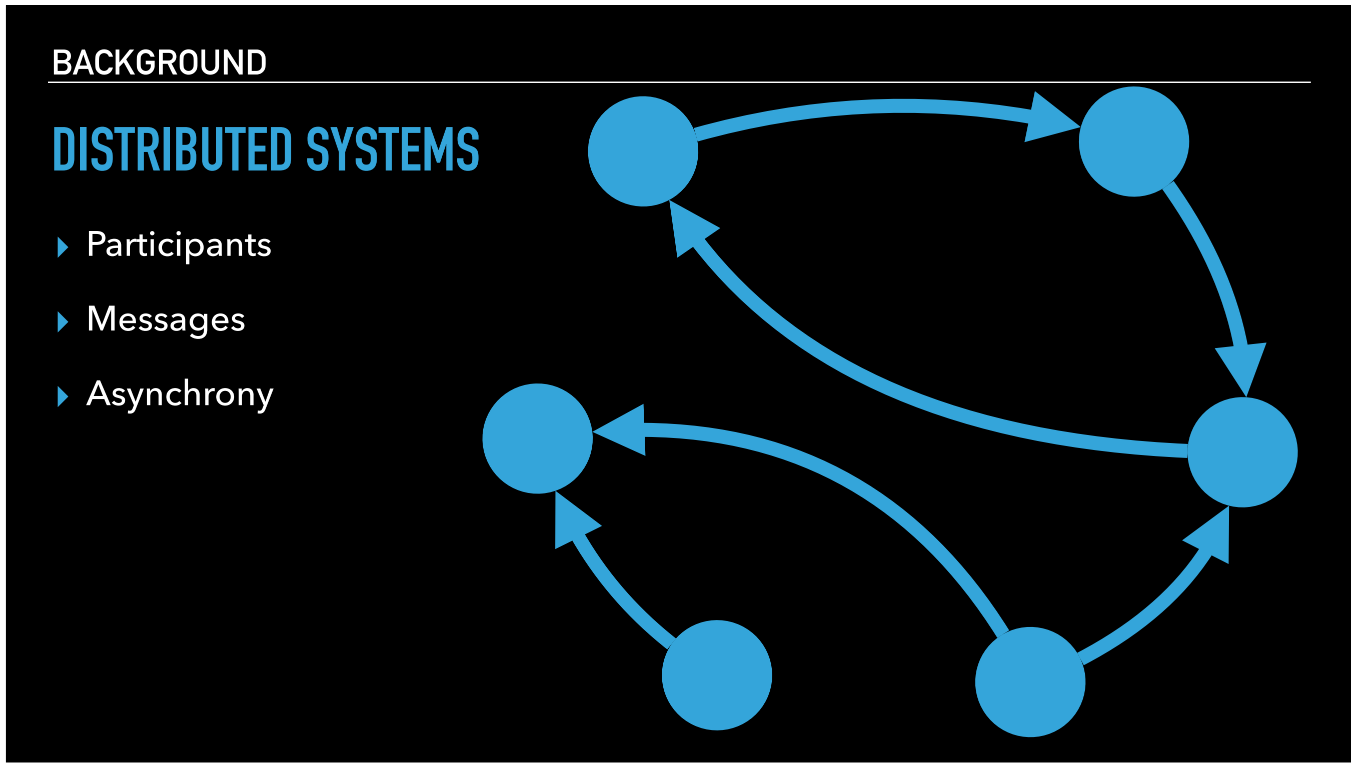
BACKGROUND

DISTRIBUTED SYSTEMS

- Participants
- Messages



Participants communicate over the network with messages to each other, which arrive eventually.
Protocols like TCP exist to ensure arrival even on lossy networks.

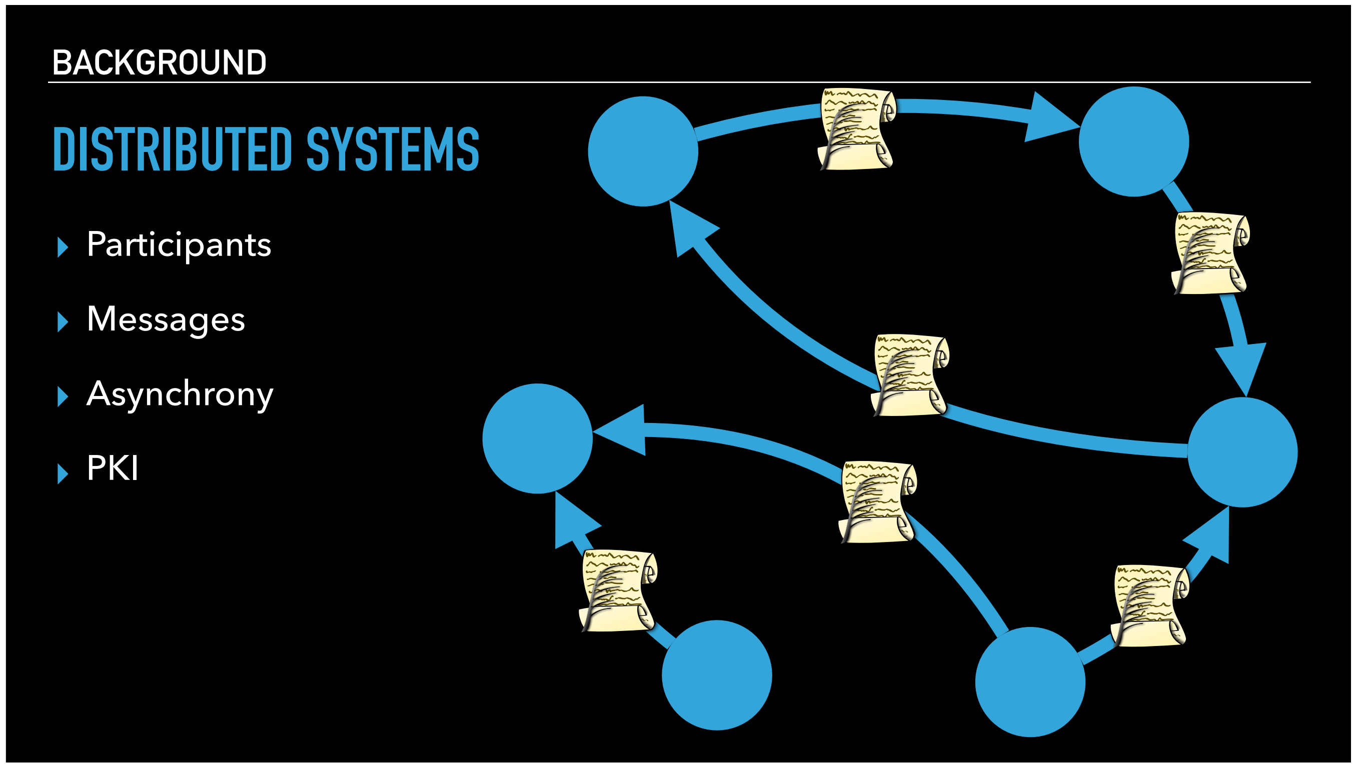


BACKGROUND

DISTRIBUTED SYSTEMS

- Participants
- Messages
- Asynchrony

However, we usually assume that there is no bound on how long messages can take.
So, these are asynchronous systems.

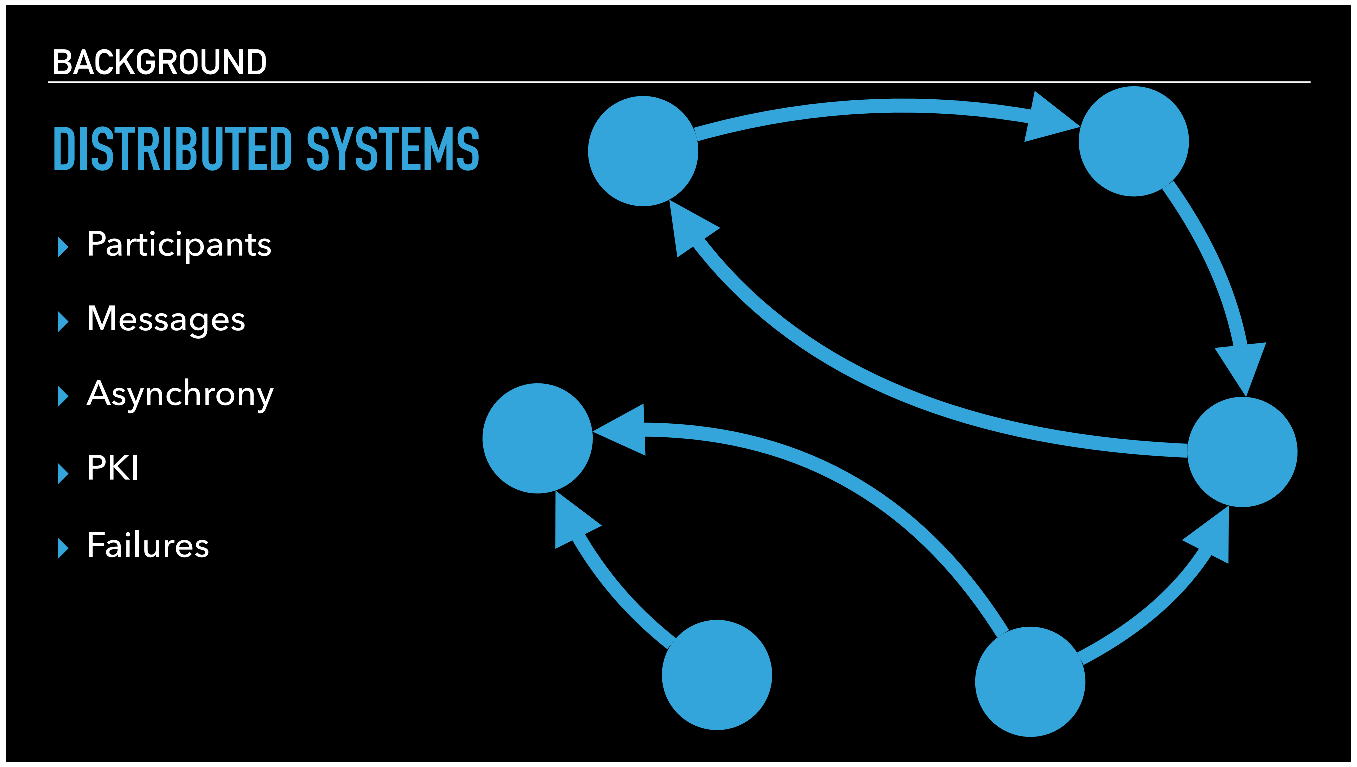


BACKGROUND

DISTRIBUTED SYSTEMS

- Participants
- Messages
- Asynchrony
- PKI

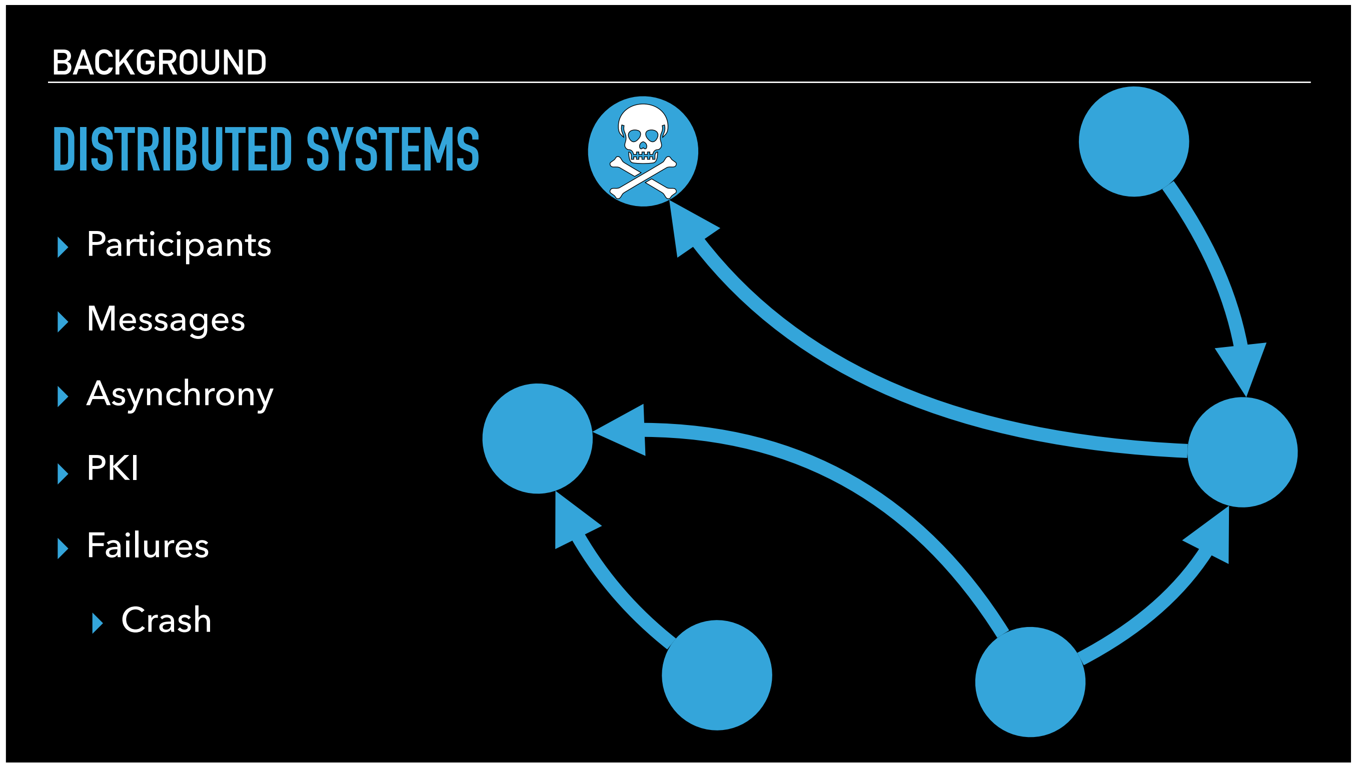
We're also going to assume that these participants have public and private keys, and most notably that they can sign messages, which cannot be forged.



DISTRIBUTED SYSTEMS

- Participants
- Messages
- Asynchrony
- PKI
- Failures

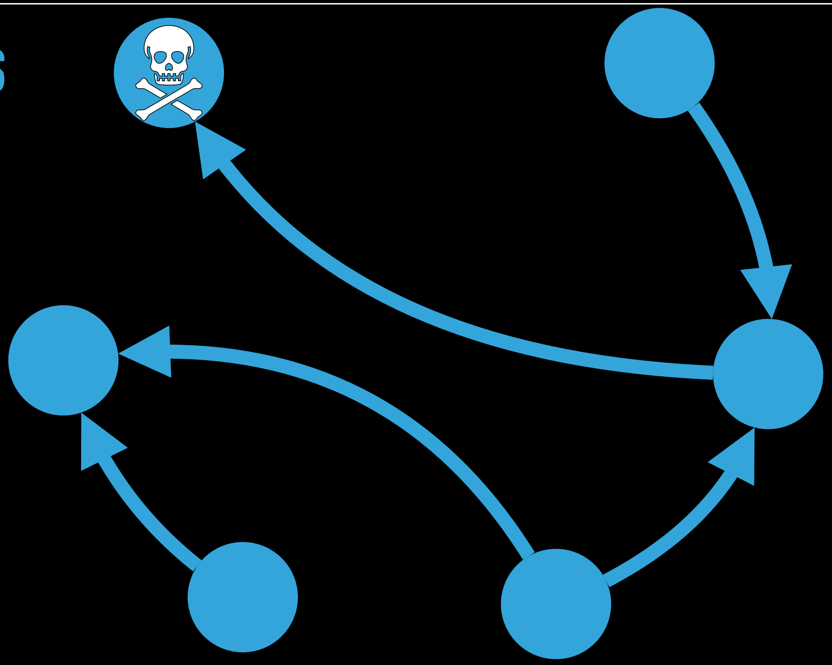
We want our distributed systems to keep working even when some participants fail.



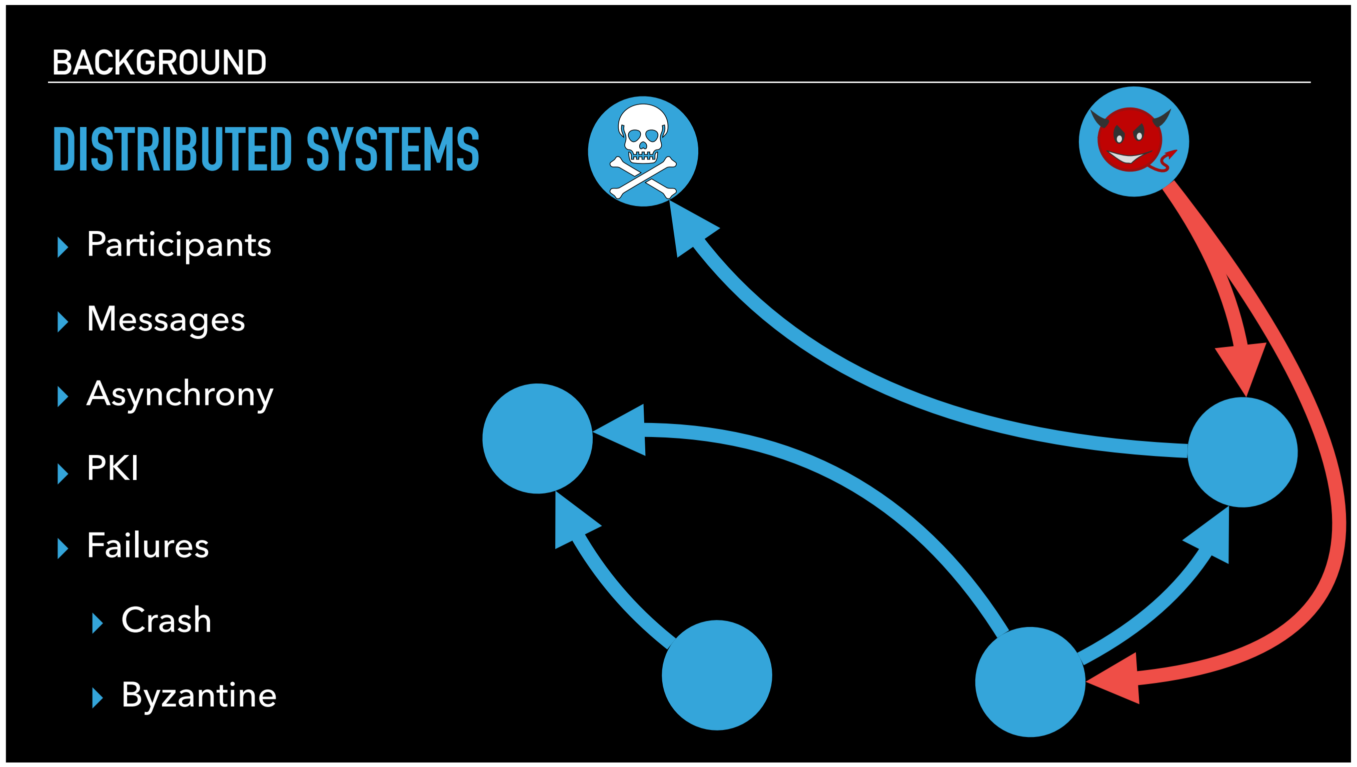
BACKGROUND

DISTRIBUTED SYSTEMS

- Participants
- Messages
- Asynchrony
- PKI
- Failures
  - Crash



Crash failures are when a node simply stops functioning, without warning or detectability.

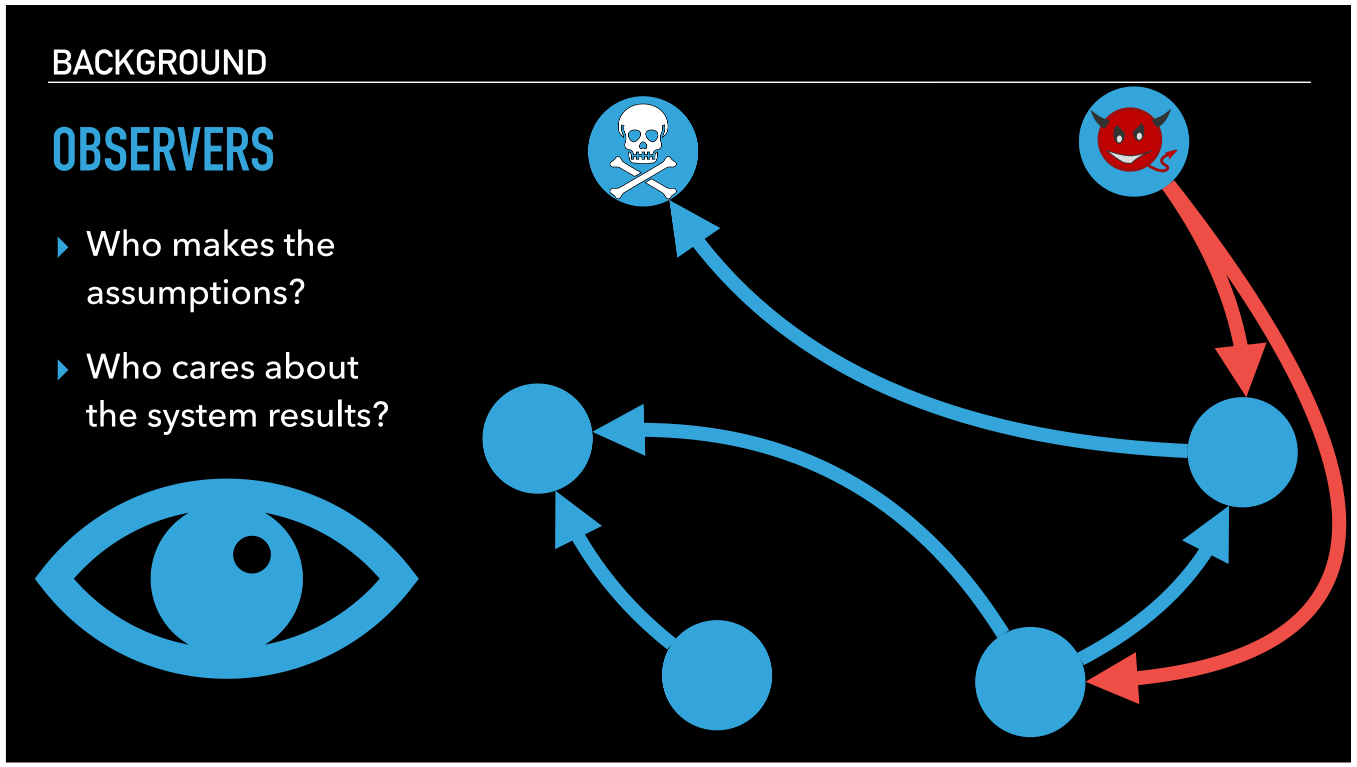


BACKGROUND

DISTRIBUTED SYSTEMS

- Participants
- Messages
- Asynchrony
- PKI
- Failures
  - Crash
  - Byzantine

Byzantine failures can behave however they want, and send messages which aren’t even part of the protocol.
They are arbitrarily evil.

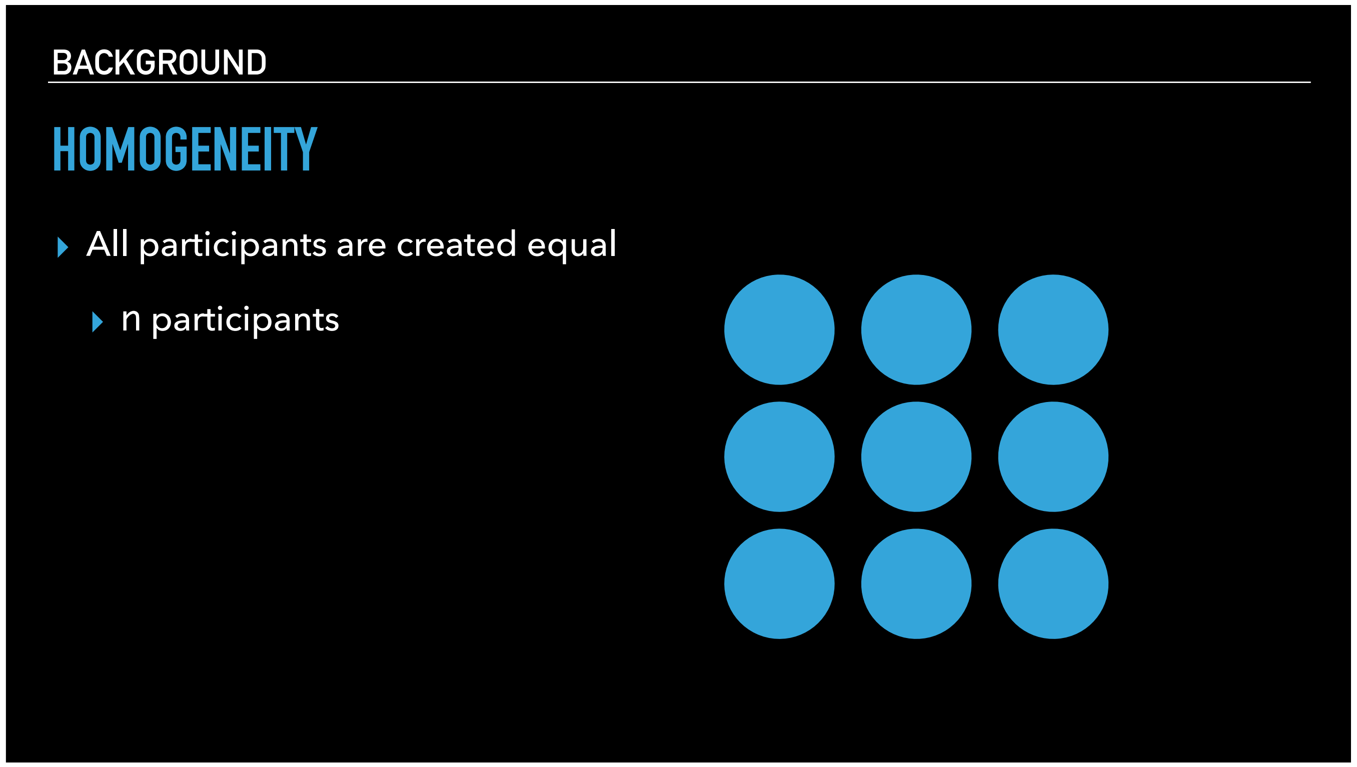


BACKGROUND

OBSERVERS

- Who makes the assumptions?
- Who cares about the system results?

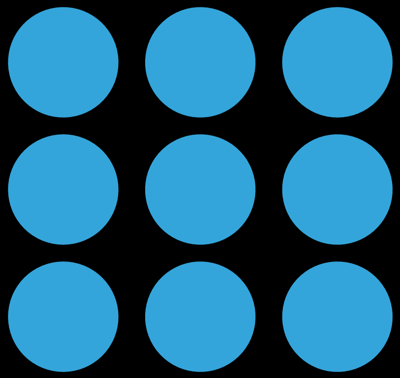
But there's one other critical component of distributed systems.
Someone observes output from the system.
Someone is making these demands about failure tolerance.
That entity is called an observer.
Typically observers are people using distributed systems.



BACKGROUND

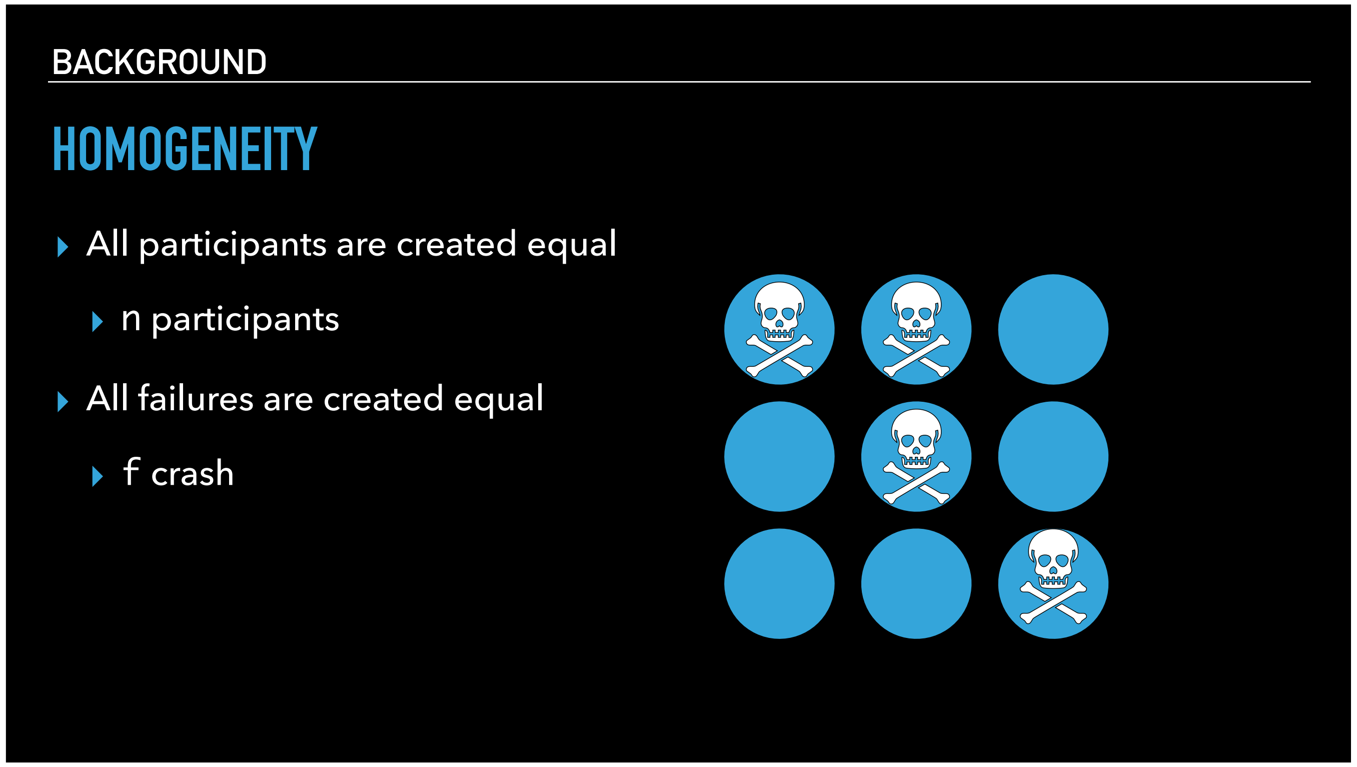
HOMOGENEITY

- All participants are created equal
  - n participants



Now, most distributed algorithms are what we call Homogeneous (as opposed to Heterogeneous), and
By that we mean 3 things:

First of all, all participants are created equal.
There may be different sets of participants with different roles, but within each role, any participant is just as good as the next.
Usually a system is characterized by having some number “n” participants.



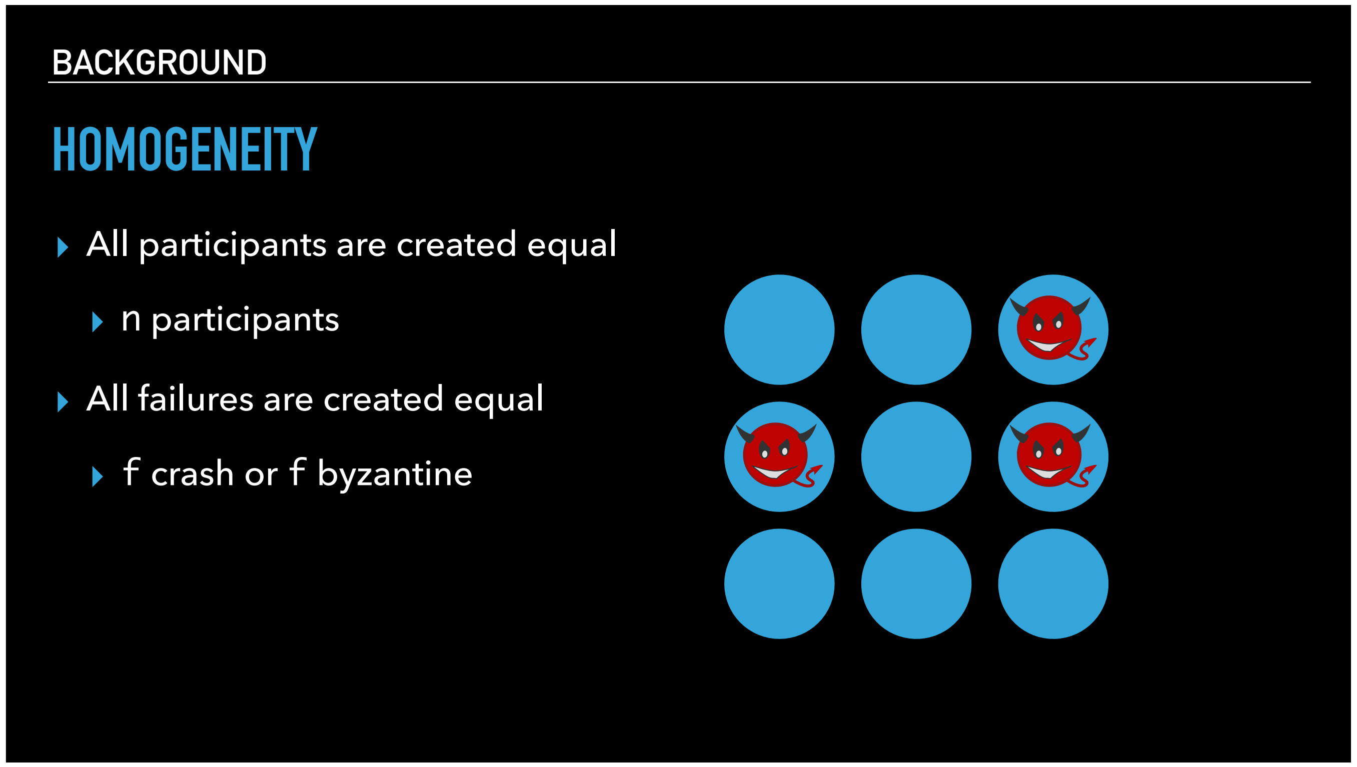
BACKGROUND

HOMOGENEITY

- All participants are created equal
  - n participants
- All failures are created equal
  - f crash

Second, all failures are created equal.
Usually, when we're analyzing a system, we talk about tolerating some number "f" failures, where they're all one type.
We'll say a system tolerates "f" crash failures, or

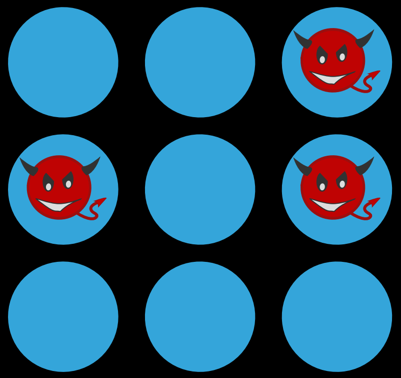
CLICK



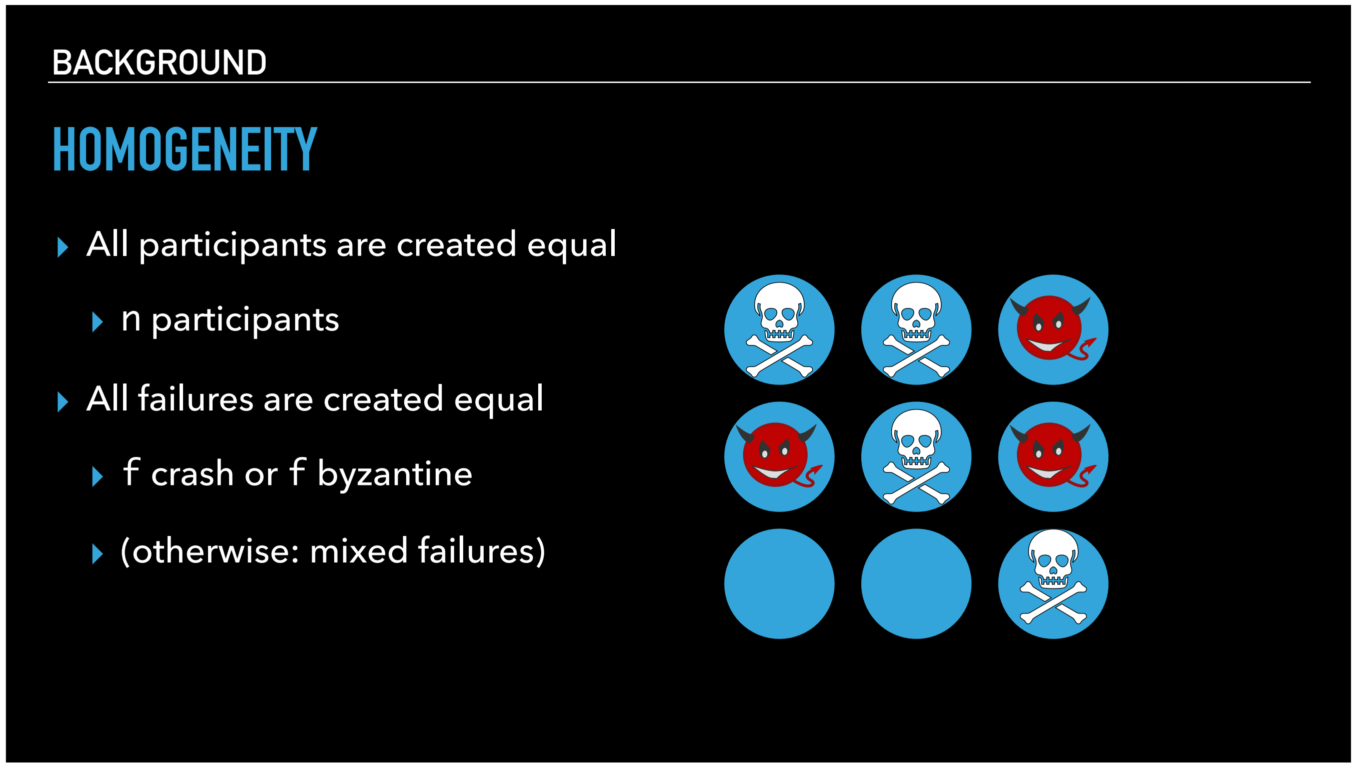
BACKGROUND

HOMOGENEITY

- All participants are created equal
  - n participants
- All failures are created equal
  - f crash or f byzantine



“f” byzantine failures

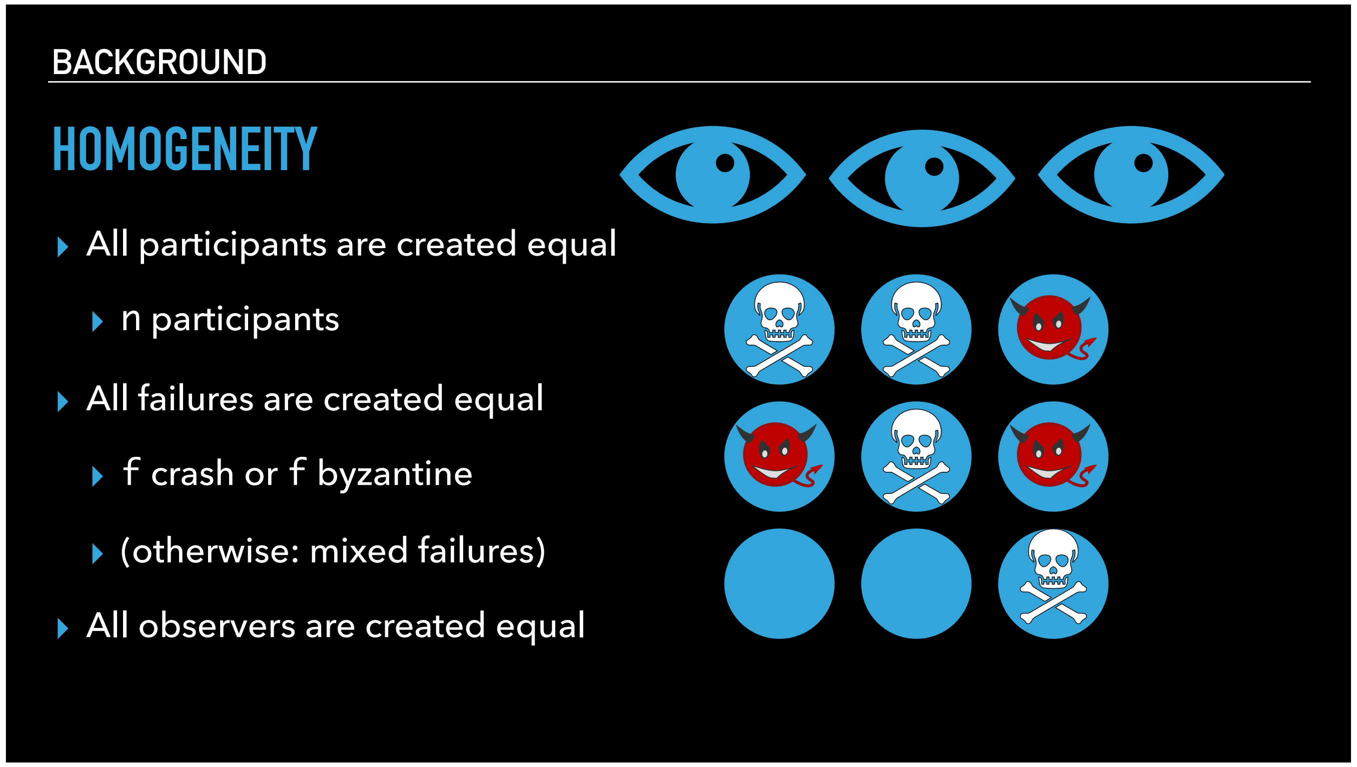


BACKGROUND

HOMOGENEITY

- All participants are created equal
  - n participants
- All failures are created equal
  - f crash or f byzantine
  - (otherwise: mixed failures)

We do sometimes talk about having some of each, in which case we call it a “mixed failure” model.

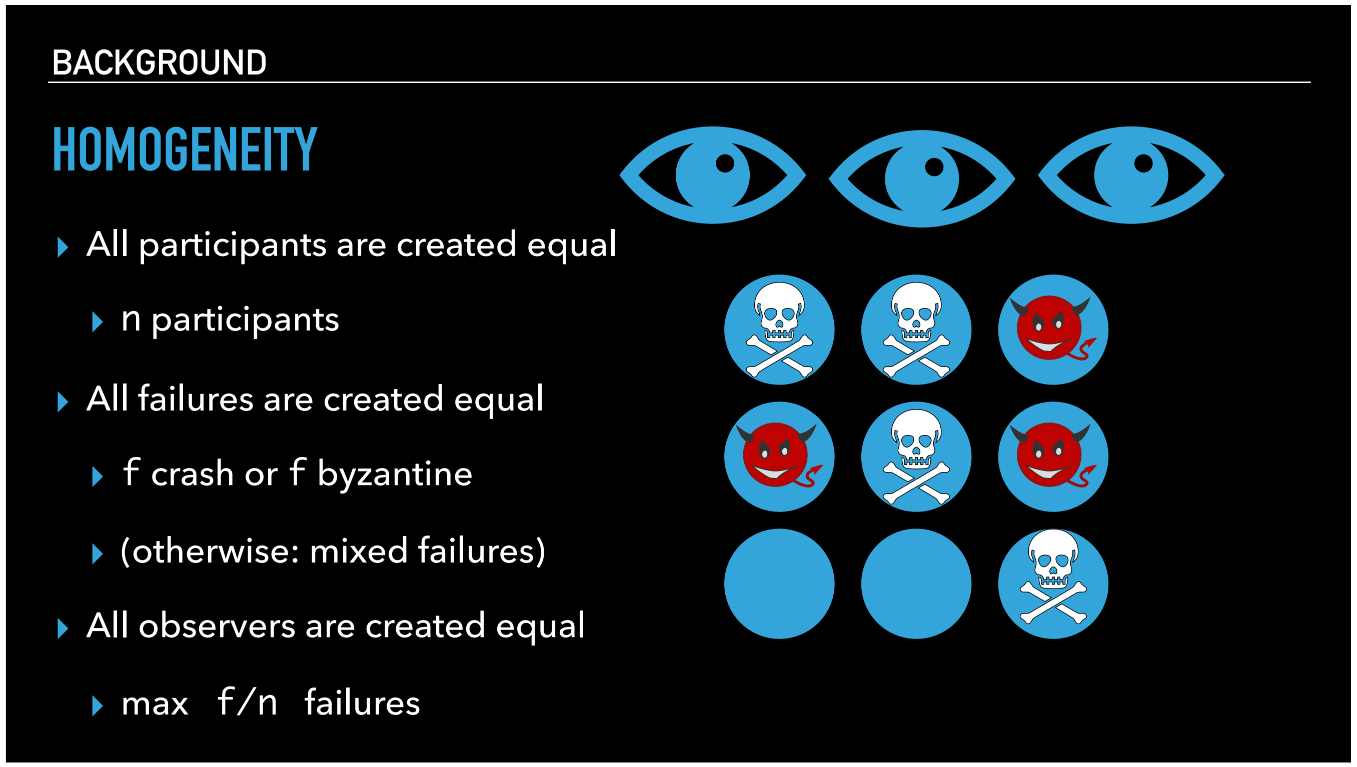


BACKGROUND

HOMOGENEITY

- All participants are created equal
  - n participants
- All failures are created equal
  - f crash or f byzantine
  - (otherwise: mixed failures)
- All observers are created equal

Third, all observers are created equal.
They all have the same assumptions about failure tolerances and results.

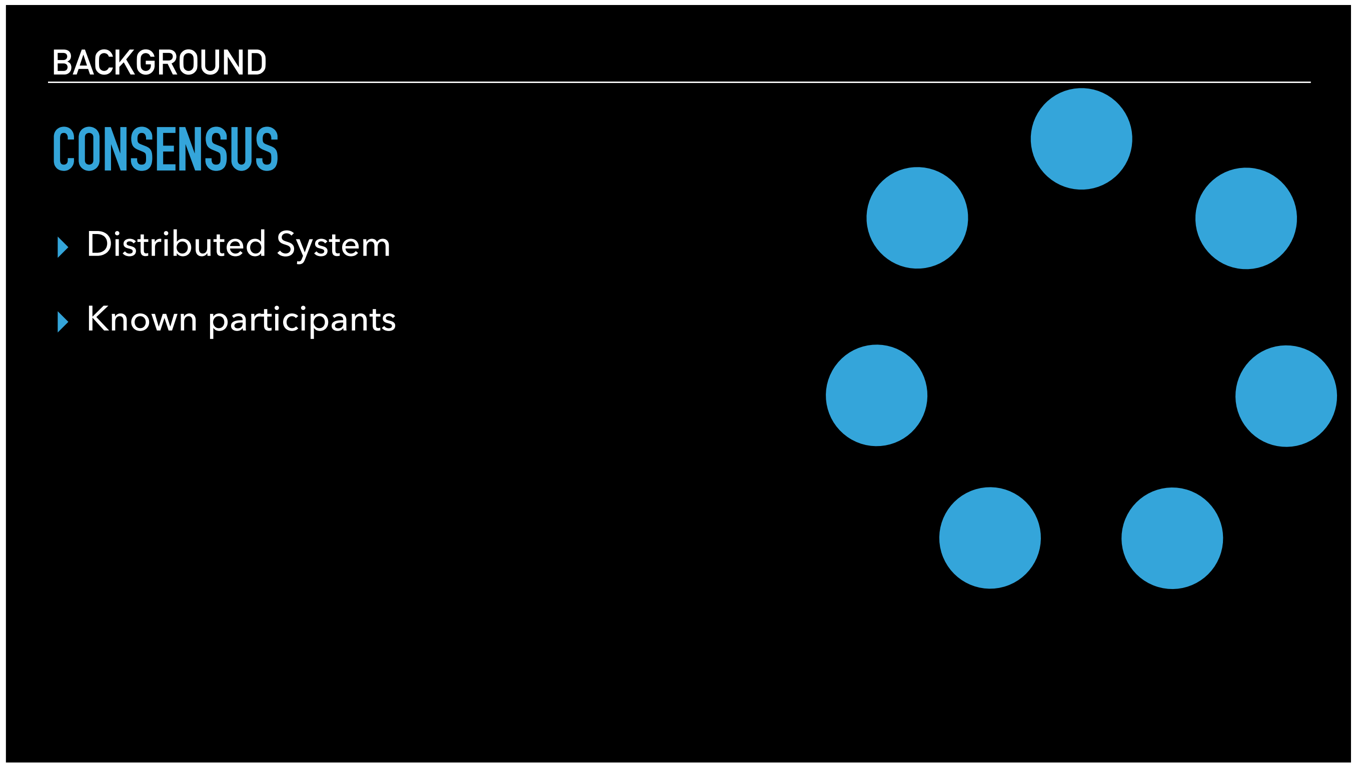


BACKGROUND

HOMOGENEITY

- All participants are created equal
  - n participants
- All failures are created equal
  - f crash or f byzantine
  - (otherwise: mixed failures)
- All observers are created equal
  - max f/n failures

Traditionally, these assumptions are something like “there will be at most f failures out of the n participants”

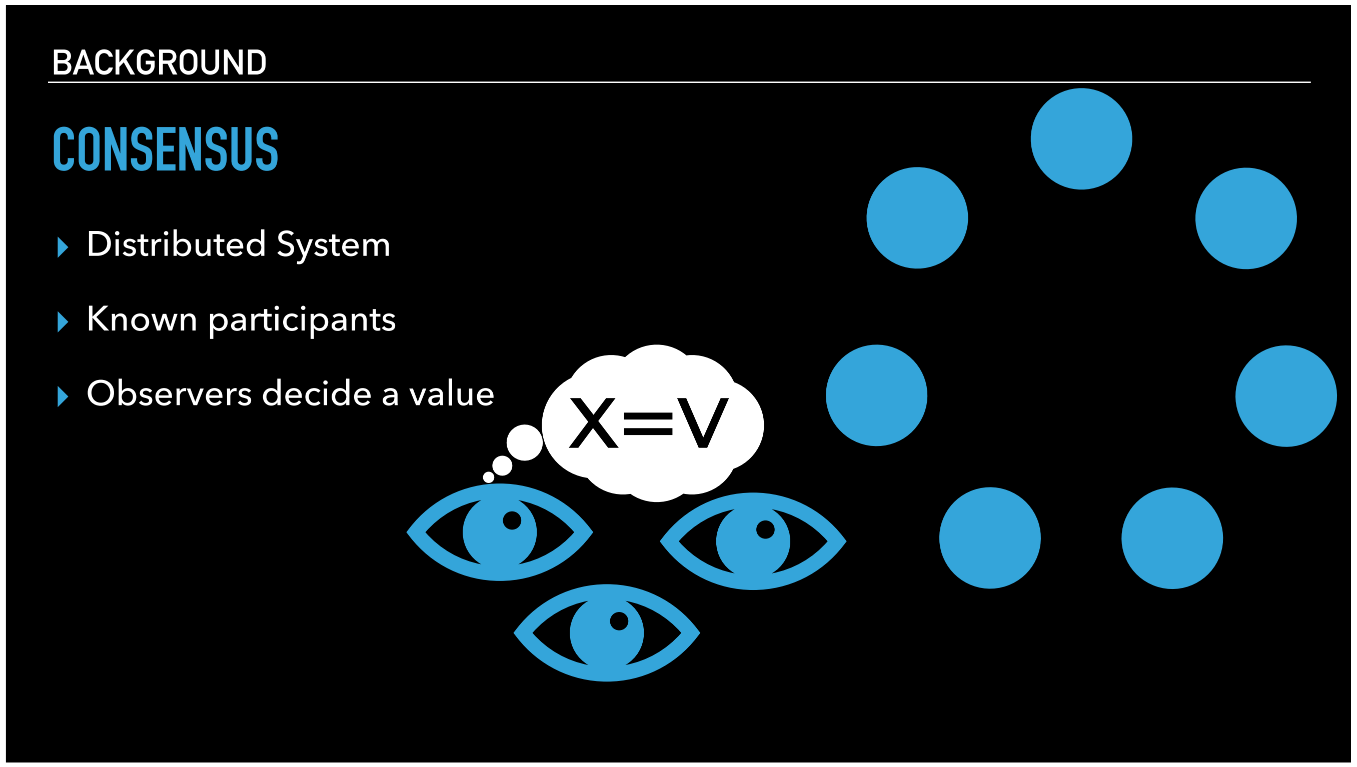


BACKGROUND

CONSENSUS

- Distributed System
- Known participants

Now, I’m here to talk about Heterogeneous Consensus, so I also have to describe what consensus means in this context.
Consensus is a task for a distributed algorithm.
We assume you have some distributed system with a known set of participants.

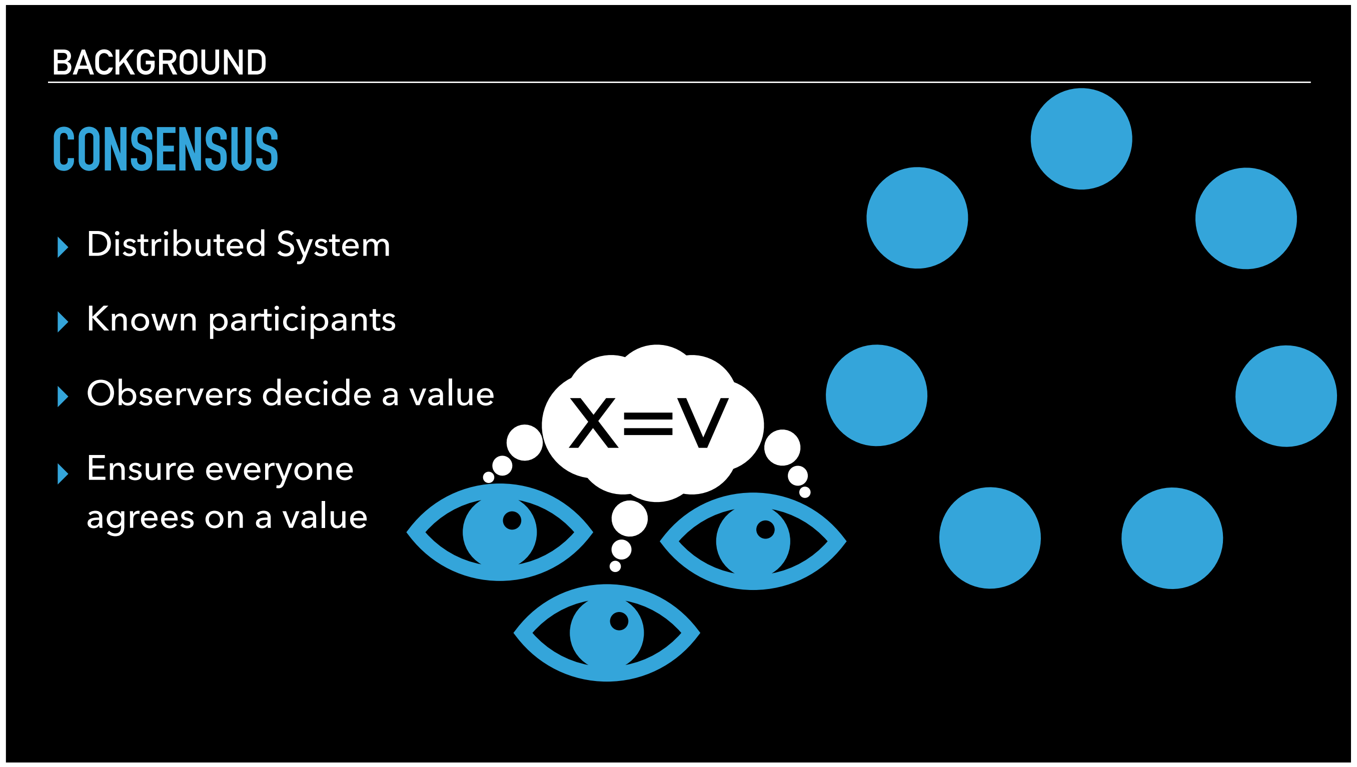


BACKGROUND

CONSENSUS

- Distributed System
- Known participants
- Observers decide a value

There are also some observers.
Each observer wants to learn, or “decide” on a value.
As Elaine pointed out on Monday, if we’re doing state machine replication, we can run consensus on each step of the state machine to simulate the machine running over time.



BACKGROUND

CONSENSUS

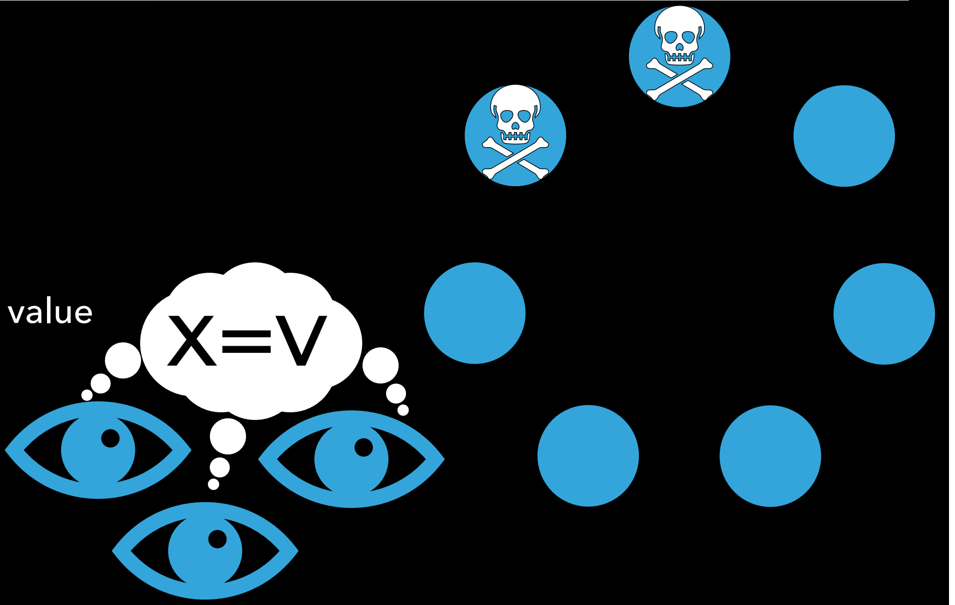
- Distributed System
- Known participants
- Observers decide a value
- Ensure everyone agrees on a value

The key property of Consensus is Agreement: we want observers to decide on the same value.

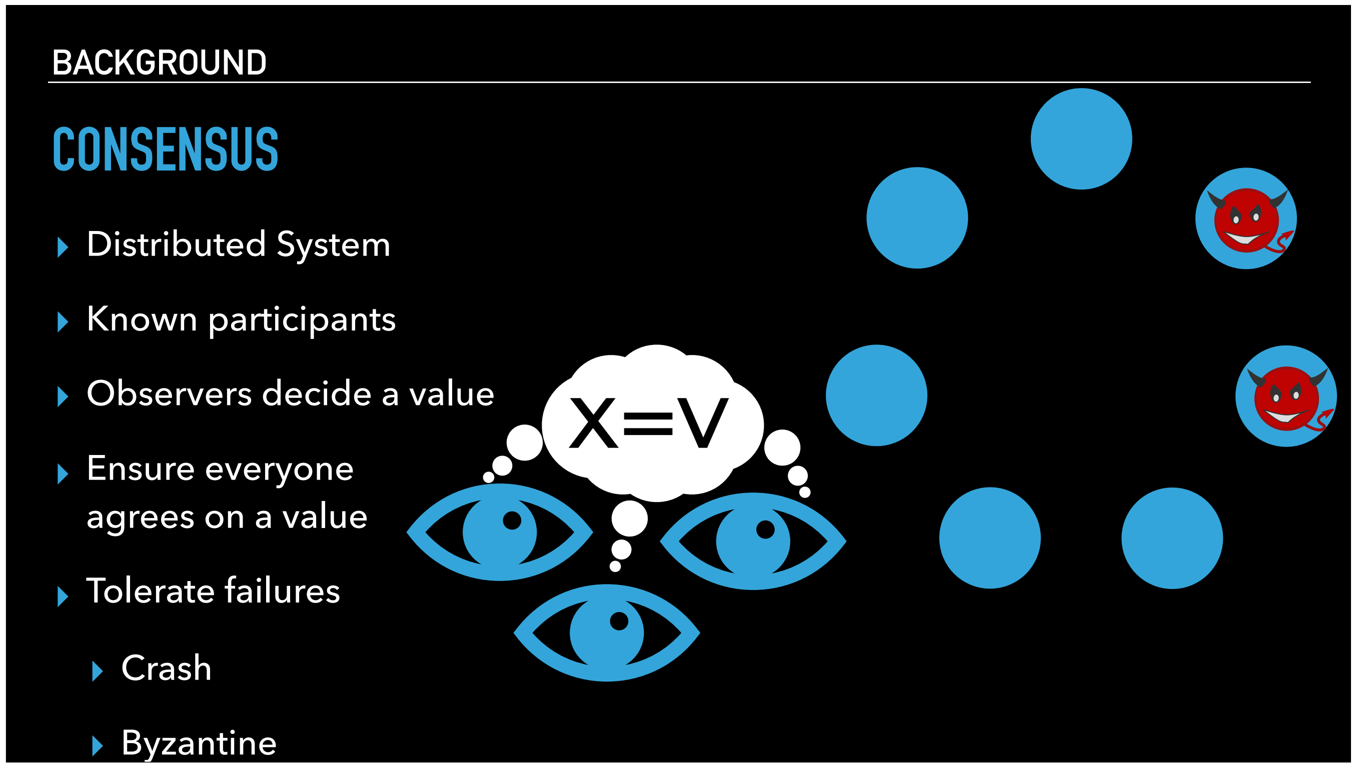
BACKGROUND

CONSENSUS

- Distributed System
- Known participants
- Observers decide a value
- Ensure everyone agrees on a value
- Tolerate failures
  - Crash



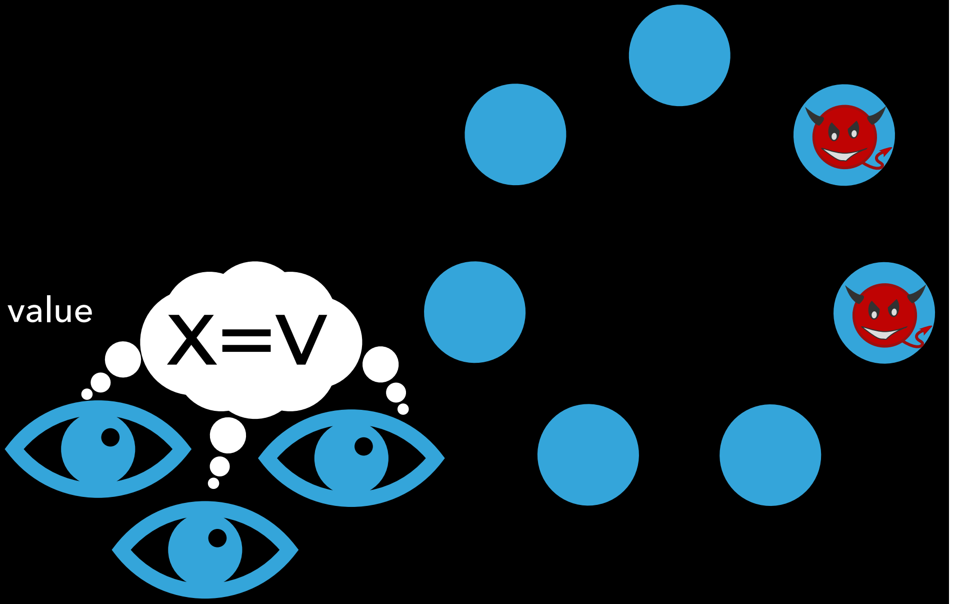
And we want this even in the presence of failures, either participants crashing



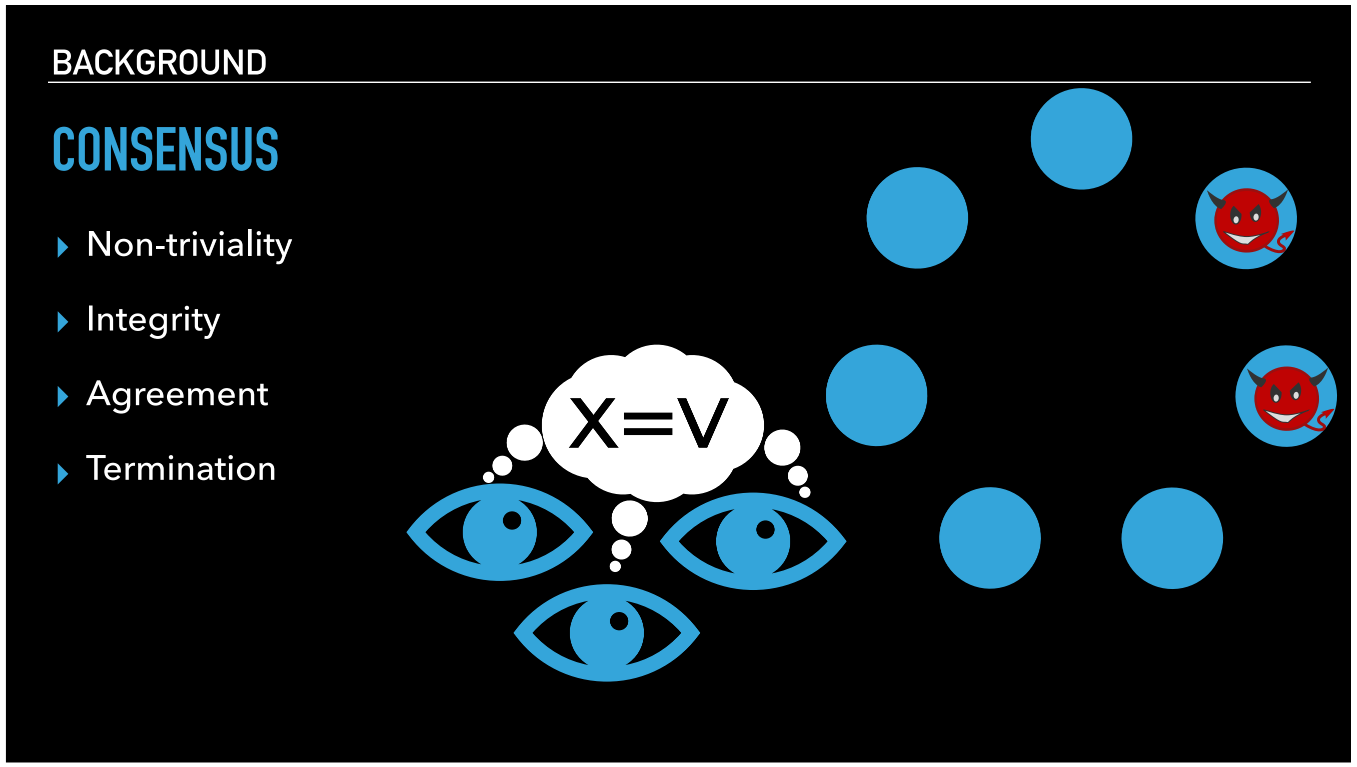
BACKGROUND

CONSENSUS

- Distributed System
- Known participants
- Observers decide a value
- Ensure everyone agrees on a value
- Tolerate failures
  - Crash
  - Byzantine

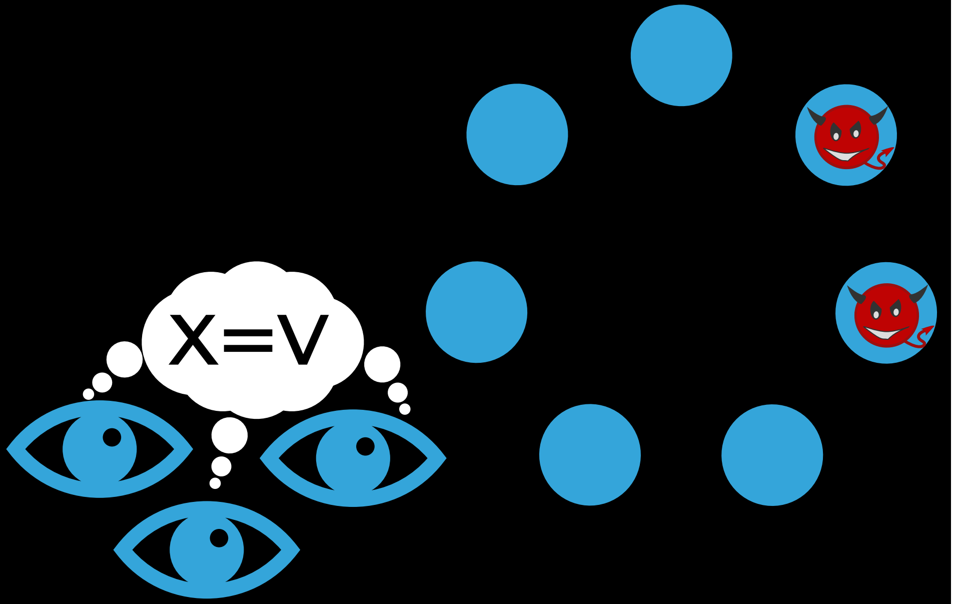


or becoming byzantine


BACKGROUND

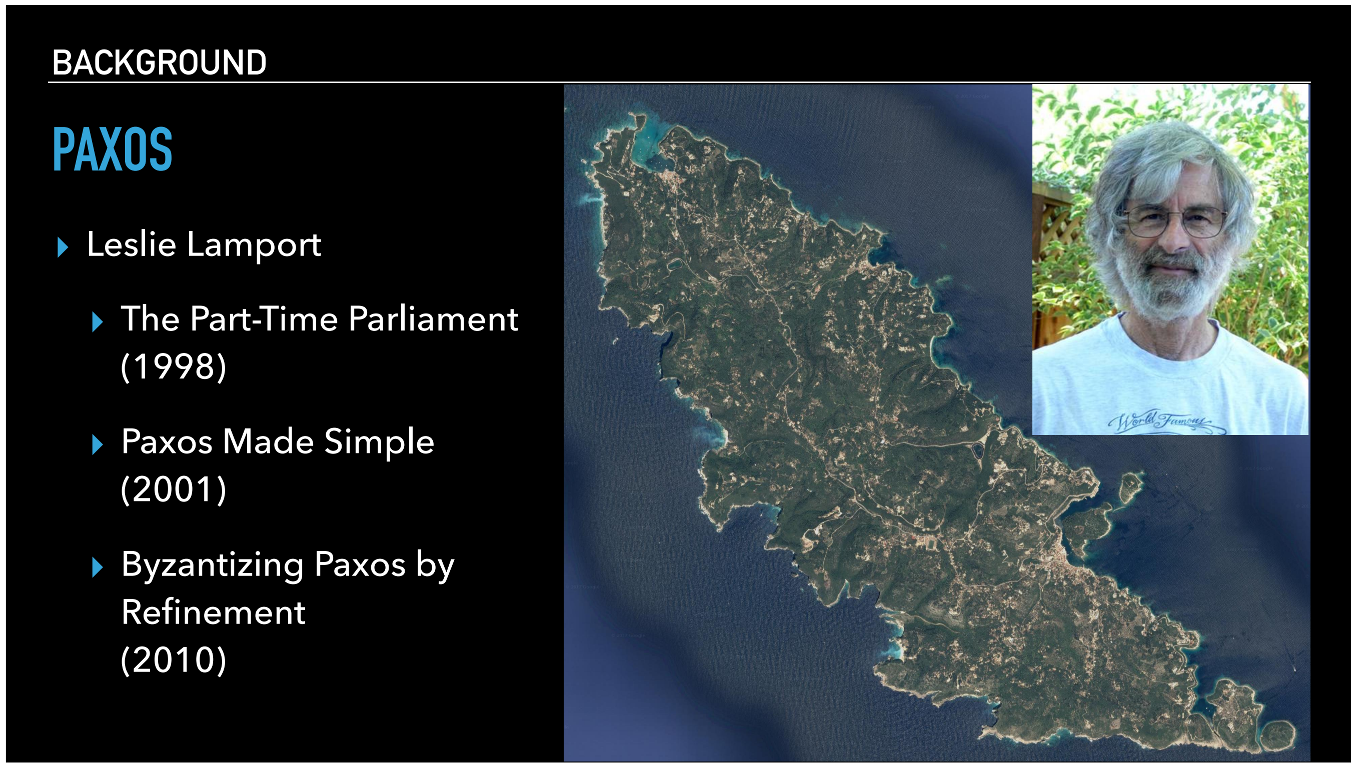
CONSENSUS

- Non-triviality
- Integrity
- Agreement
- Termination



Formally, Consensus algorithms need 4 properties:
Non-triviality: any value decided by an observer wa initially “proposed” by some party authorized to do so.
This is why you can’t always just decide 3.
Integrity: an observer decides at most once.
Agreement: So long as your failure assumptions hold, if two observers both decide, they decide the same value
Termination: So long as your failure assumptions hold, each observer decides eventually.

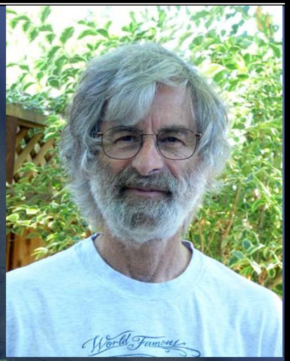
There’s a famous proof that you can’t have all 4, but we can usually prove the first 3, and then get Termination if you assume partially synchronous messages.



BACKGROUND

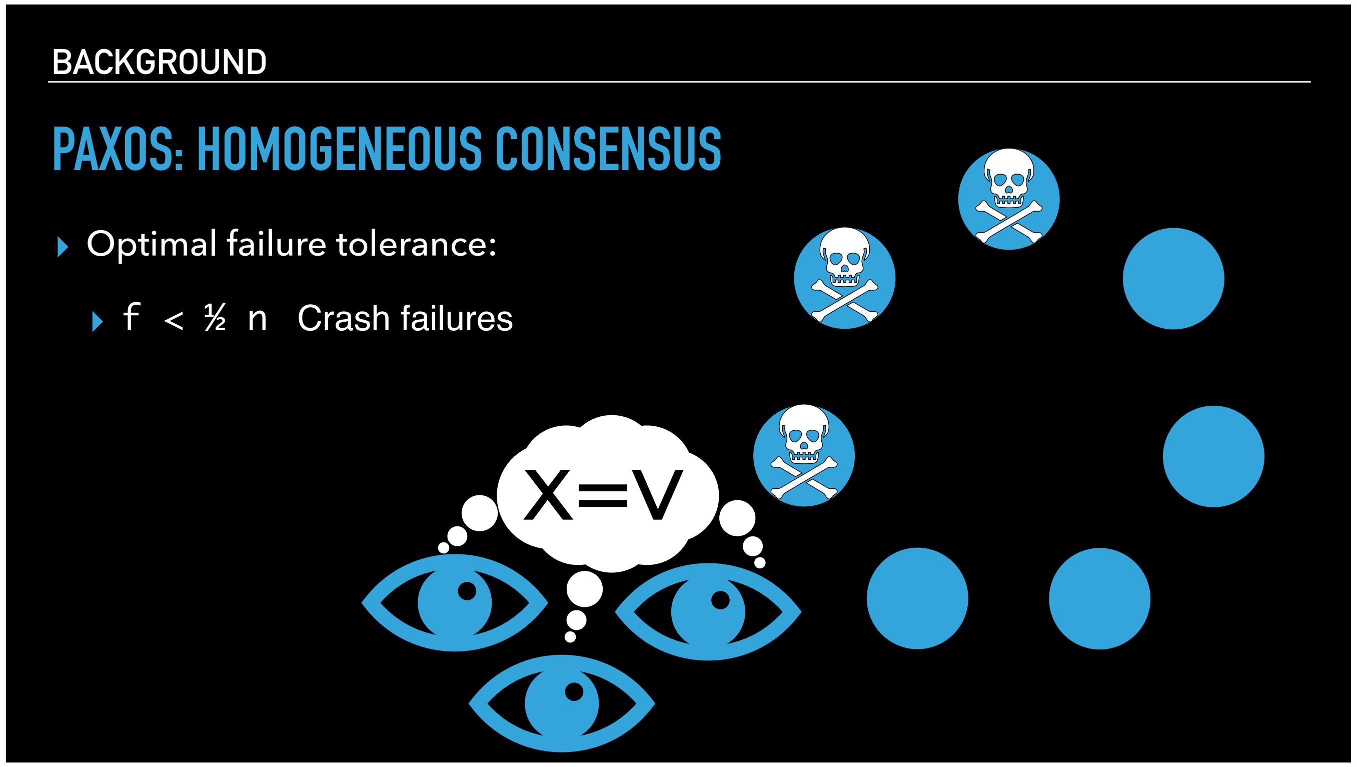
PAXOS

- Leslie Lamport
  - The Part-Time Parliament (1998)
  - Paxos Made Simple (2001)
  - Byzantizing Paxos by Refinement (2010)



The most famous consensus algorithm is Paxos, named after this Island, by Leslie Lamport.
He first published it in 1998, but that paper was too confusing, so
he published “Paxos Made Simple” in 2001, which is slightly less confusing.

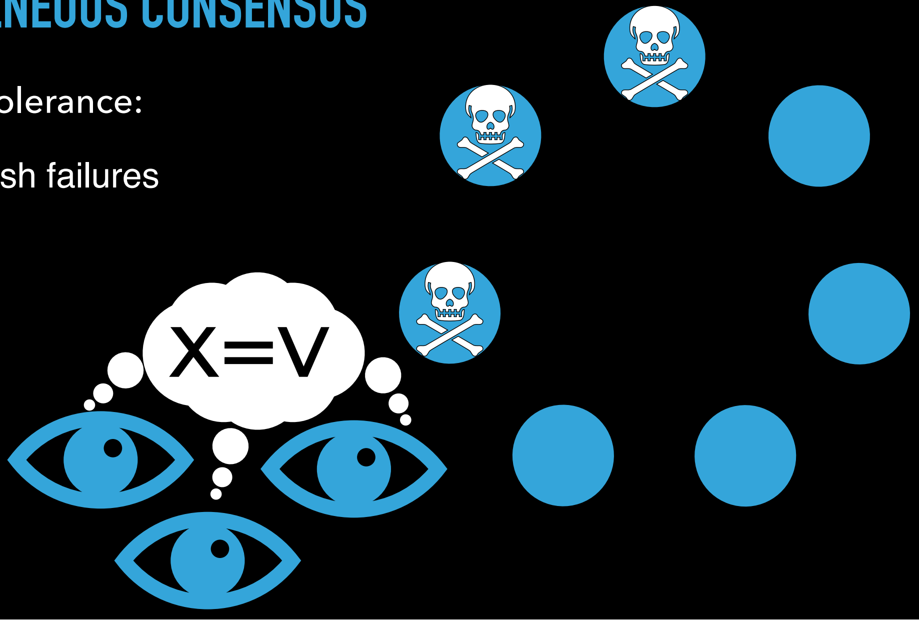
The original Paxos tolerates only Crash failures, but there have been several variants designed to deal with Byzantine failures as well.
The one we’re going to use is the protocol from Lamport’s “Byzantizing Paxos by Refinement.”



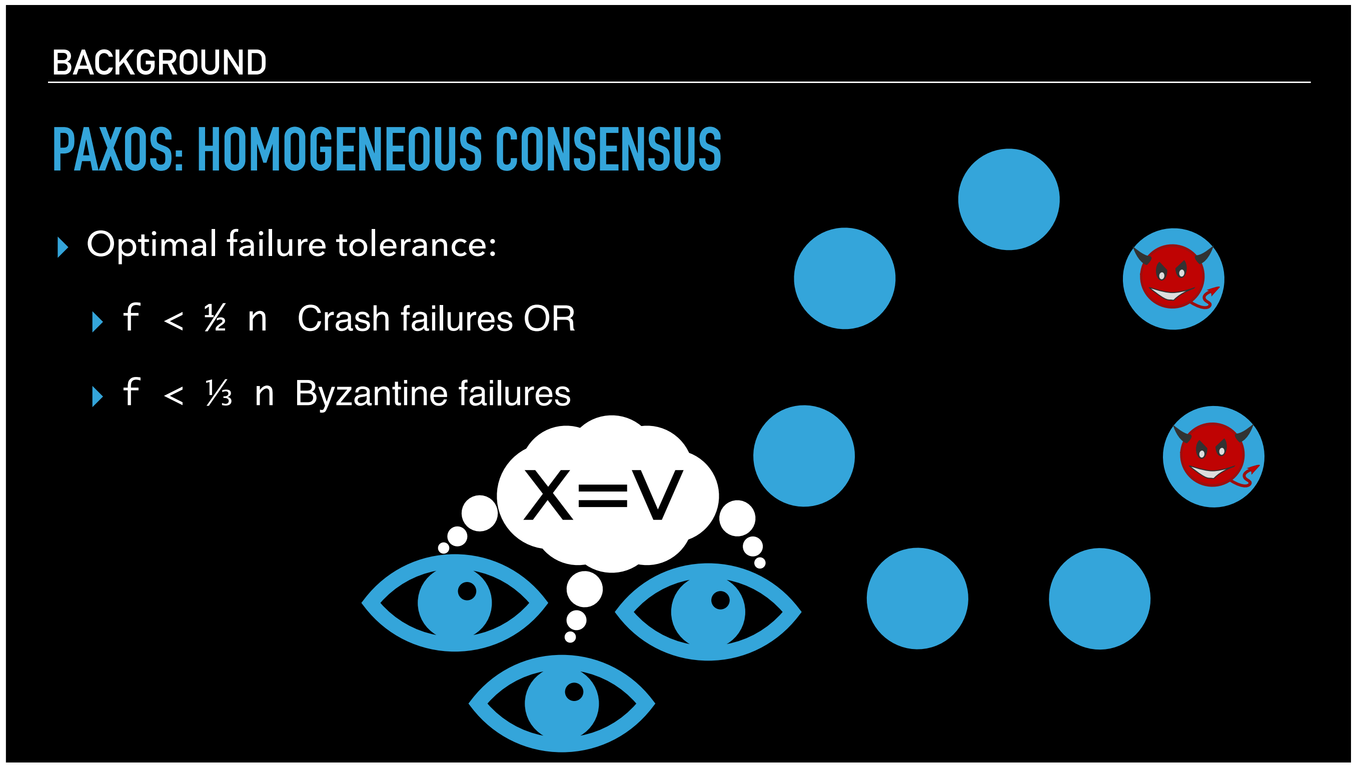
BACKGROUND

PAXOS: HOMOGENEOUS CONSENSUS

- Optimal failure tolerance:
  - $f < \frac{1}{2} n$ Crash failures



So Paxos, viewed as a Homogenous Consensus algorithm,
Can be proven to have optimal failure tolerance.
You can make a Paxos instance tolerate up to f Crash failures, where f is strictly < half the number of participants.

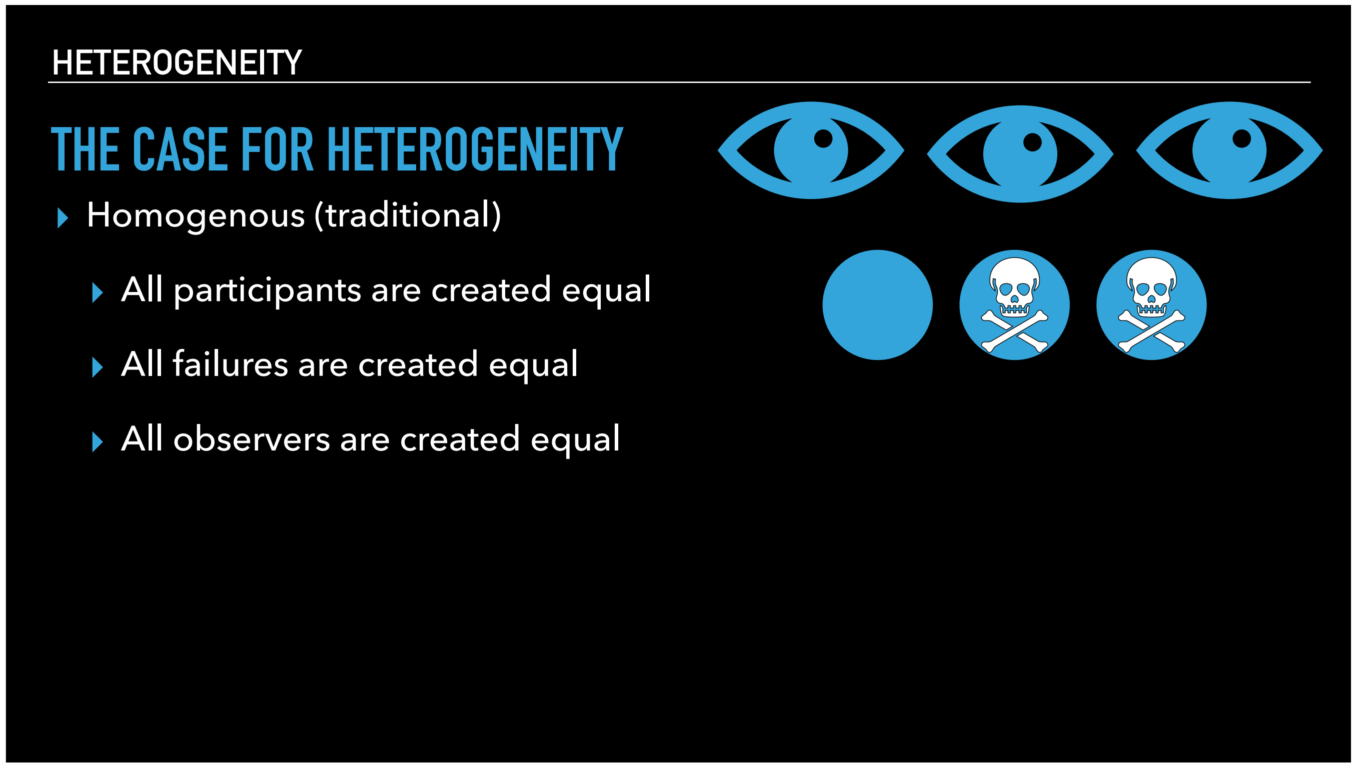


BACKGROUND

PAXOS: HOMOGENEOUS CONSENSUS

- Optimal failure tolerance:
  - $f < \frac{1}{2} n$ Crash failures OR
  - $f < \frac{1}{3} n$ Byzantine failures

Alternatively,
You can make a Paxos instance tolerate up to f Byzantine failures, where f is strictly < a third the number of participants.
This is also proven to be optimal.



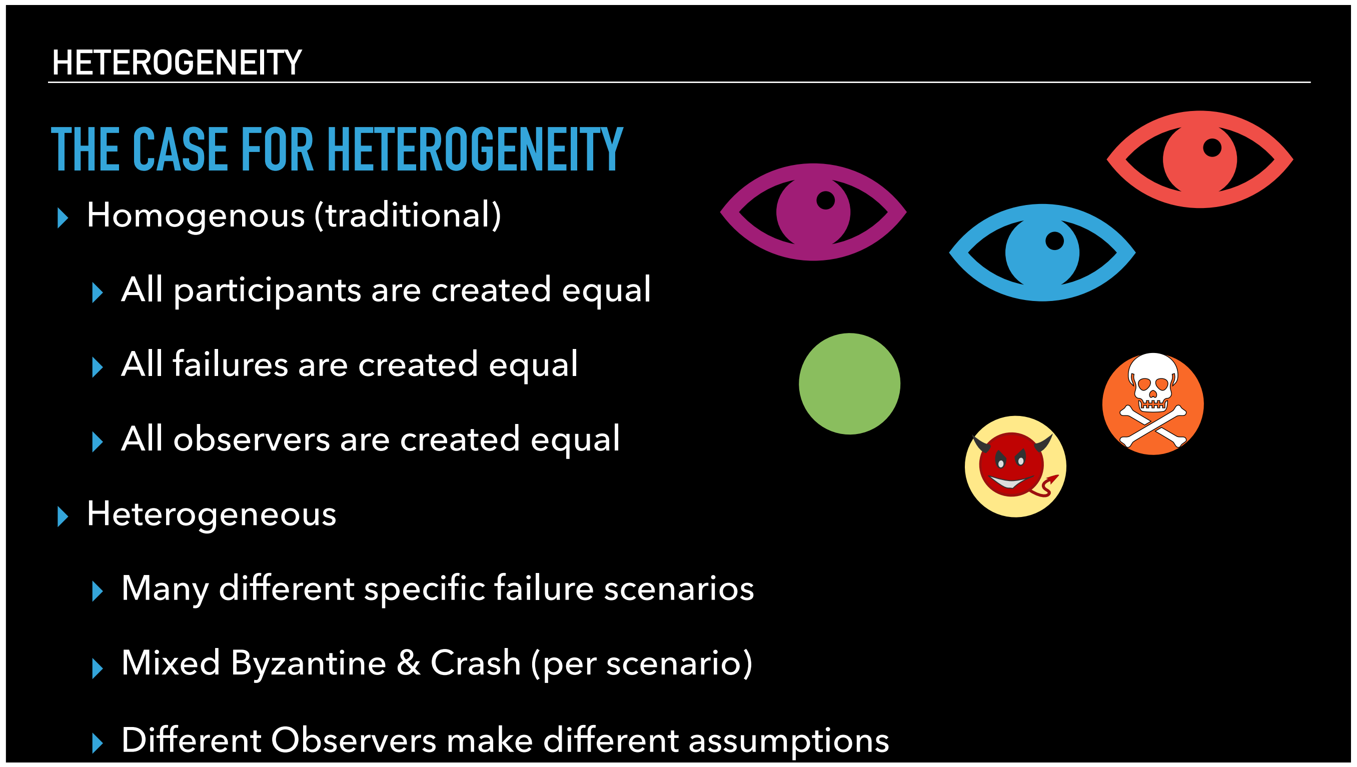
HETEROGENEITY

THE CASE FOR HETEROGENEITY

- Homogenous (traditional)
  - All participants are created equal
  - All failures are created equal
  - All observers are created equal

So that's Paxos as a homogeneous algorithm.
As a reminder, that means
  - All participants are created equal
  - All failures are created equal
  - all observers are created equal

But I'm here to tell you that we don't always want to operate under Homogeneous assumptions

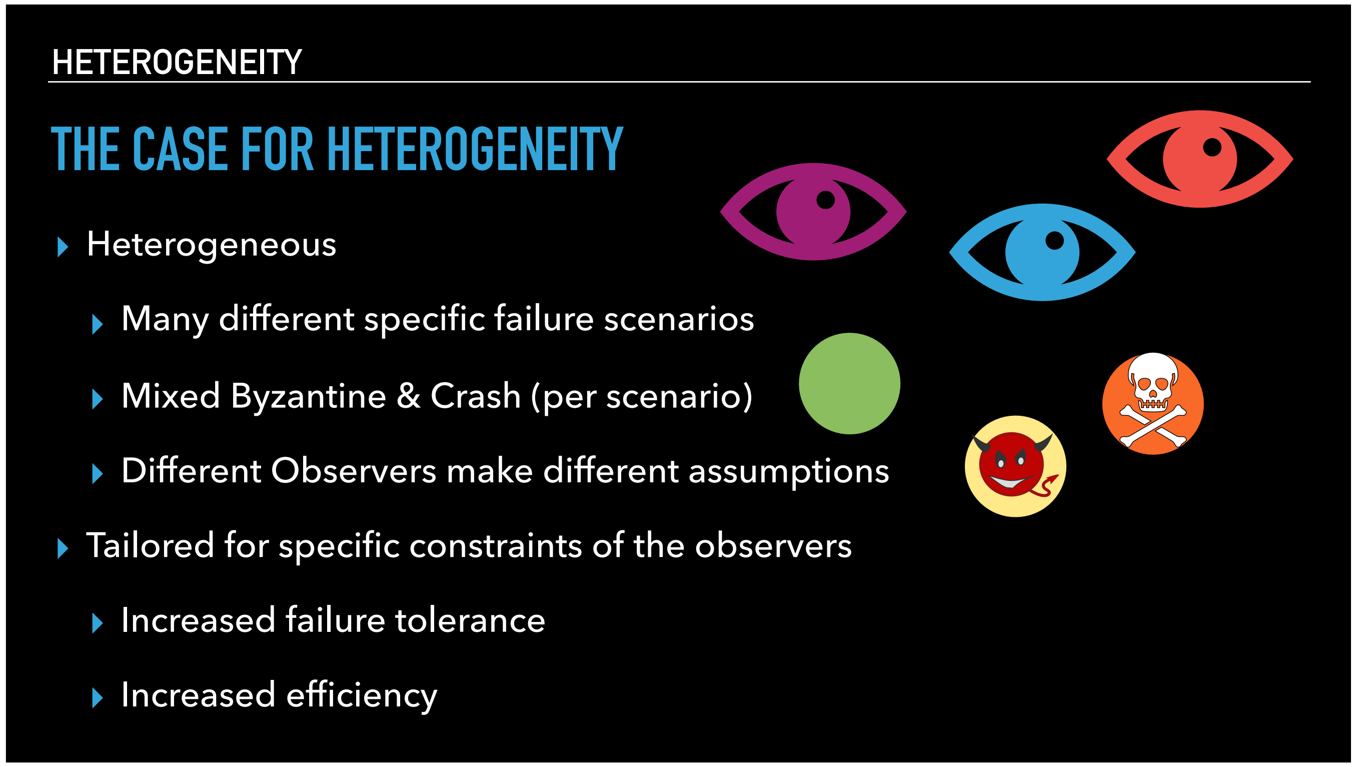


HETEROGENEITY

THE CASE FOR HETEROGENEITY

- Homogenous (traditional)
  - All participants are created equal
  - All failures are created equal
  - All observers are created equal
- Heterogeneous
  - Many different specific failure scenarios
  - Mixed Byzantine & Crash (per scenario)
  - Different Observers make different assumptions

Sometimes we want Heterogeneous systems.
These are systems with many different, not-necessarily-symmetric, tolerated failure conditions,
where there may be some crash failures and some Byzantine failures tolerated at the same time, and
where different observers make different assumptions about what may happen, and what guarantees they want under which conditions.



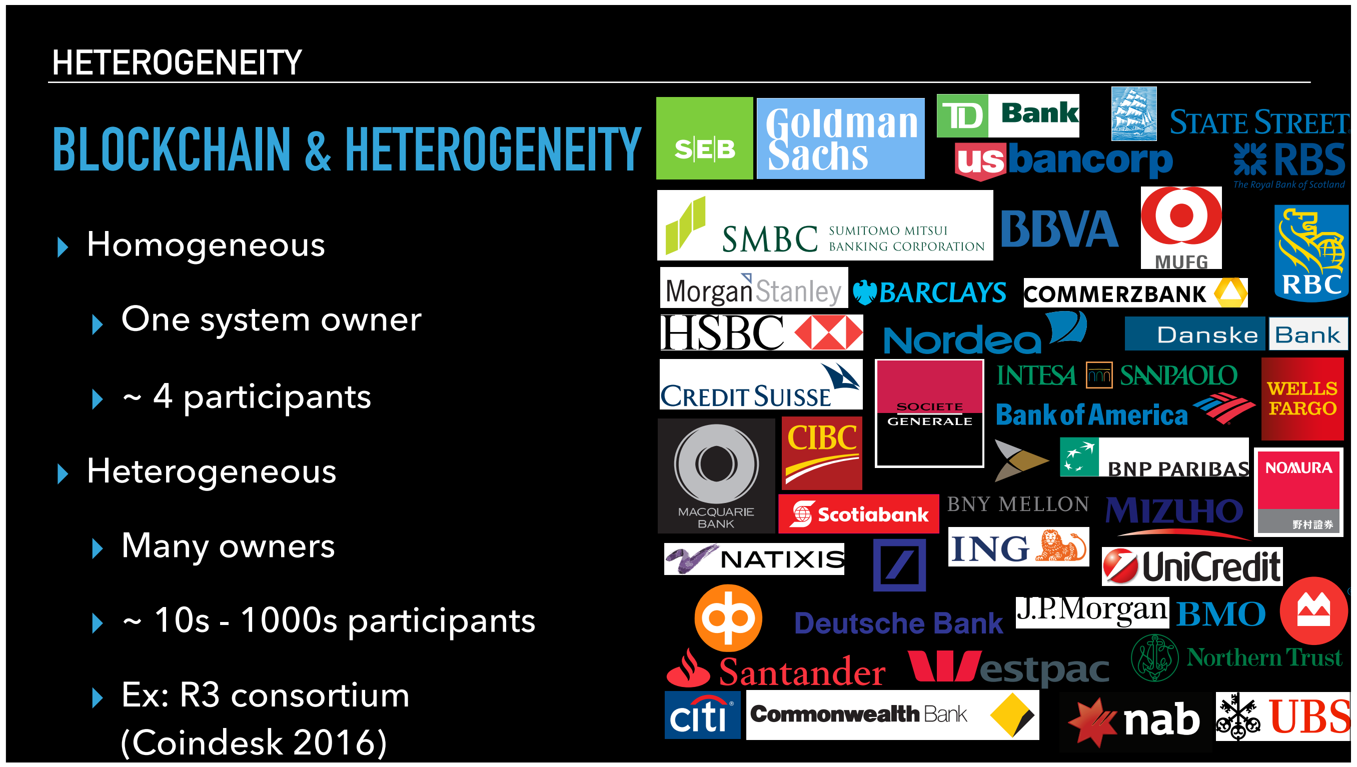
HETEROGENEITY

THE CASE FOR HETEROGENEITY

- Heterogeneous
  - Many different specific failure scenarios
  - Mixed Byzantine & Crash (per scenario)
  - Different Observers make different assumptions
- Tailored for specific constraints of the observers
  - Increased failure tolerance
  - Increased efficiency

Heterogeneity means that we can tailor a system to the specific constraints of the observers
and the potential failures of the participants.

In the case where everybody really is the same, we want our Heterogeneous System to reduce to the Homogeneous version, but otherwise,
we should be able to tolerate *more* failures,
be *more* efficient with our resources.



HETEROGENEITY

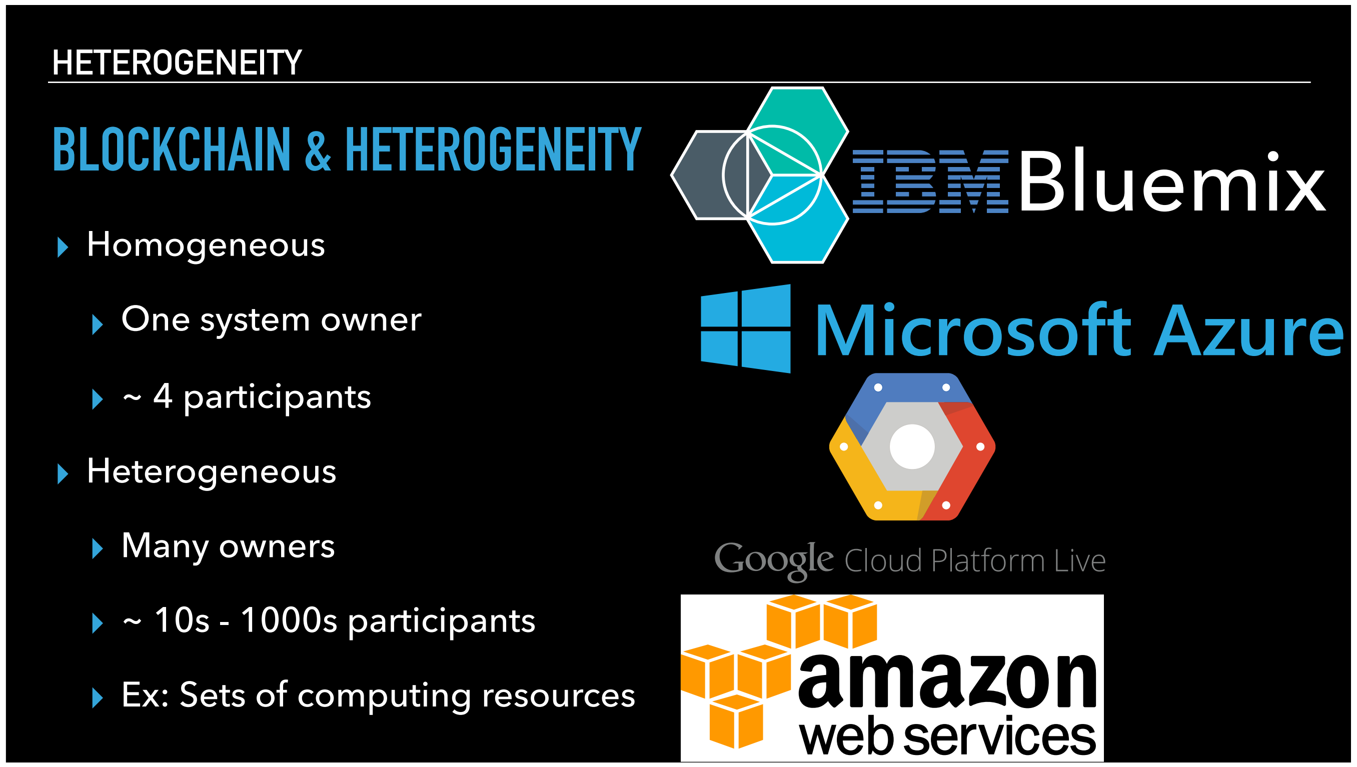
BLOCKCHAIN & HETEROGENEITY

- Homogeneous
  - One system owner
  - ~ 4 participants
- Heterogeneous
  - Many owners
  - ~ 10s - 1000s participants
  - Ex: R3 consortium (Coindesk 2016)

So why should a Blockchain audience care about Heterogeneity?

Well, most Homogeneous systems run under one Owner, or one Company.
Consensus systems typically have 3 or 4 participants, either geo-distributed or in one data-center.

In contrast, in permissioned Blockchains, we see systems with many owners,
with tens to thousands of participants.
For instance, These are the 42 members of R3's blockchain consortium as of some time last year.
(there are more now)
They don't necessarily all agree on the failure demands of the system or trustworthiness of its many participants.



HETEROGENEITY

BLOCKCHAIN & HETEROGENEITY

- Homogeneous
  - One system owner
  - ~ 4 participants
- Heterogeneous
  - Many owners
  - ~ 10s - 1000s participants
  - Ex: Sets of computing resources

IBM Bluemix

Microsoft Azure

Google Cloud Platform Live

amazon web services

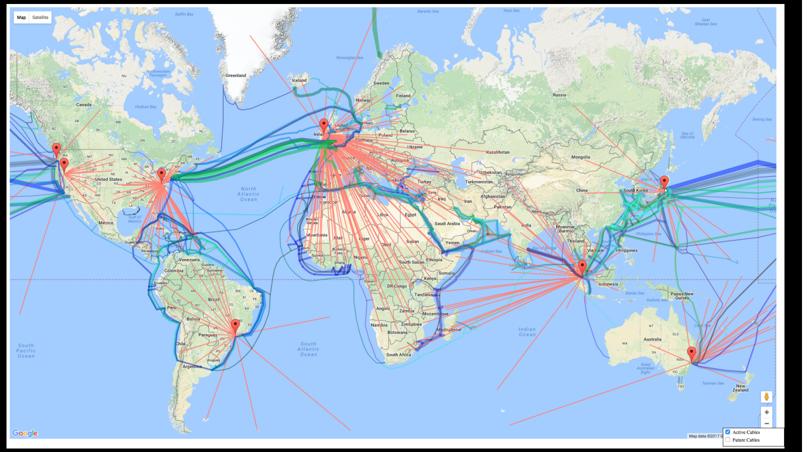
Blockchains can have participant servers running under many different roofs,
All these companies can be used to run blockchain participants, in addition to the servers owned by the companies on the last slide.

Not every observer is going to have the same opinion about the trustworthiness of participants in each of these services.

HETEROGENEITY

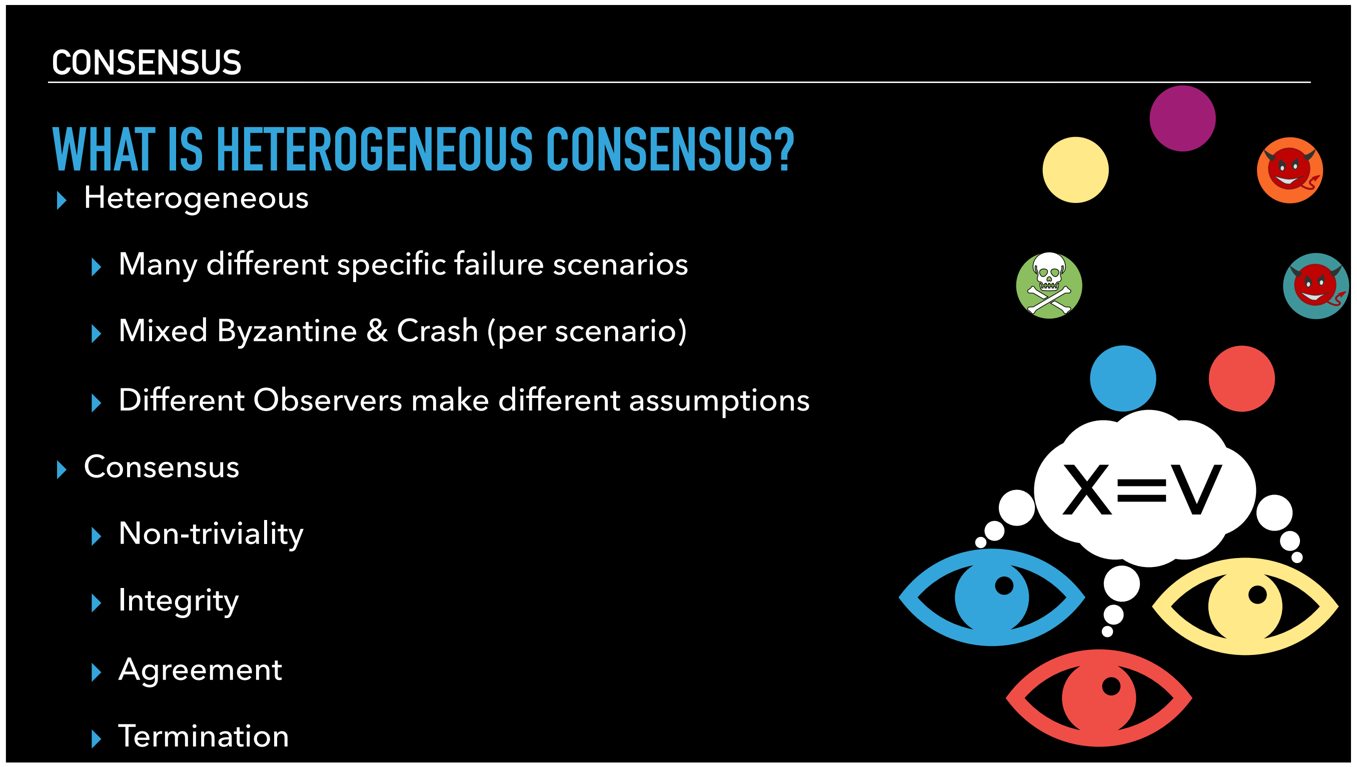
BLOCKCHAIN & HETEROGENEITY

- Homogeneous
  - One system owner
  - ~ 4 participants
- Heterogeneous
  - Many owners
  - ~ 10s - 1000s participants
  - Ex: Correlated failures



AWS Datacenters used by TurnKey Linux
(http://turnkeylinux.github.io/aws-datacenters/)

And not all failures are independent.
Participants sharing an implementation are more likely to be hacked and go byzantine together.
Participants in the same data center or power grid are more likely to crash together.
Participants owned by the same people may fail together.
Correlated failures happen all the time, so its not always an accurate threat model to tolerate “any f independent failures.”



CONSENSUS

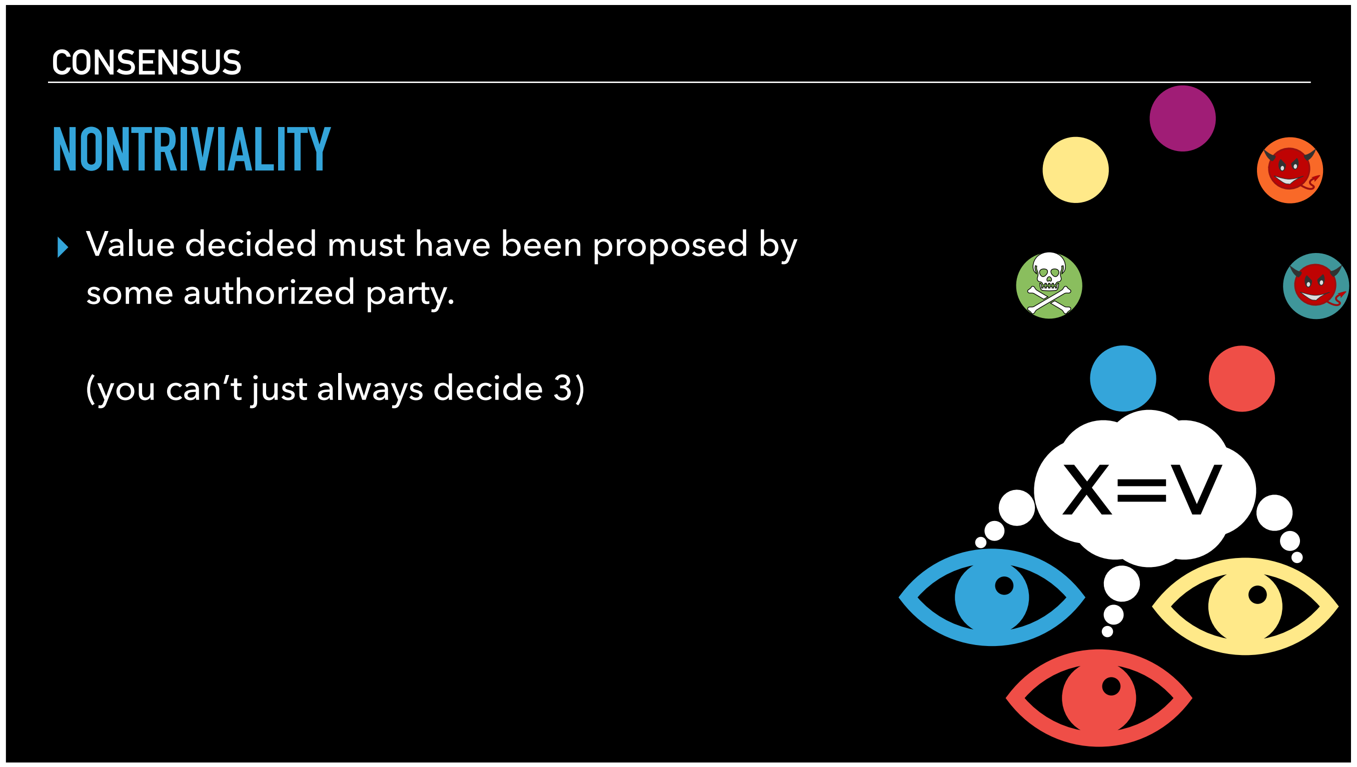
WHAT IS HETEROGENEOUS CONSENSUS?

- Heterogeneous
  - Many different specific failure scenarios
  - Mixed Byzantine & Crash (per scenario)
  - Different Observers make different assumptions
- Consensus
  - Non-triviality
  - Integrity
  - Agreement
  - Termination

OK, so what does it take to have a Heterogeneous Consensus algorithm.
Well, it has to be Heterogeneous, that means we have
- Many different failure scenarios
- Mixed Byzantine and Crash failures
- Different observers make different assumptions

And it has to be consensus, so it has
- Non-triviality
- Integrity
- Agreement
- Termination

Let’s take a closer look at what those Consensus properties mean in a Heterogeneous setting.

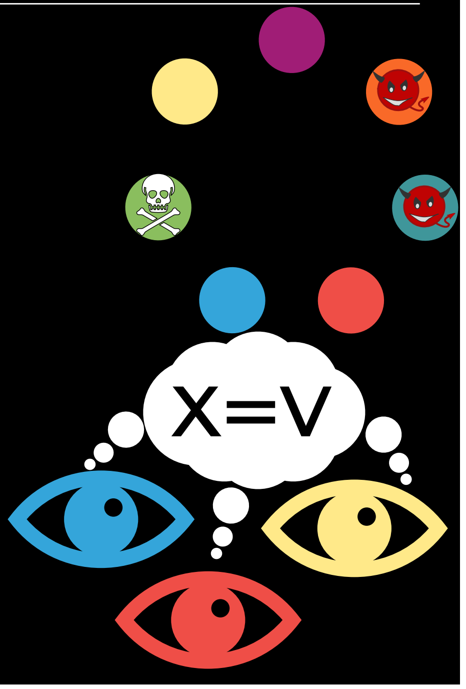


CONSENSUS

NONTRIVIALITY

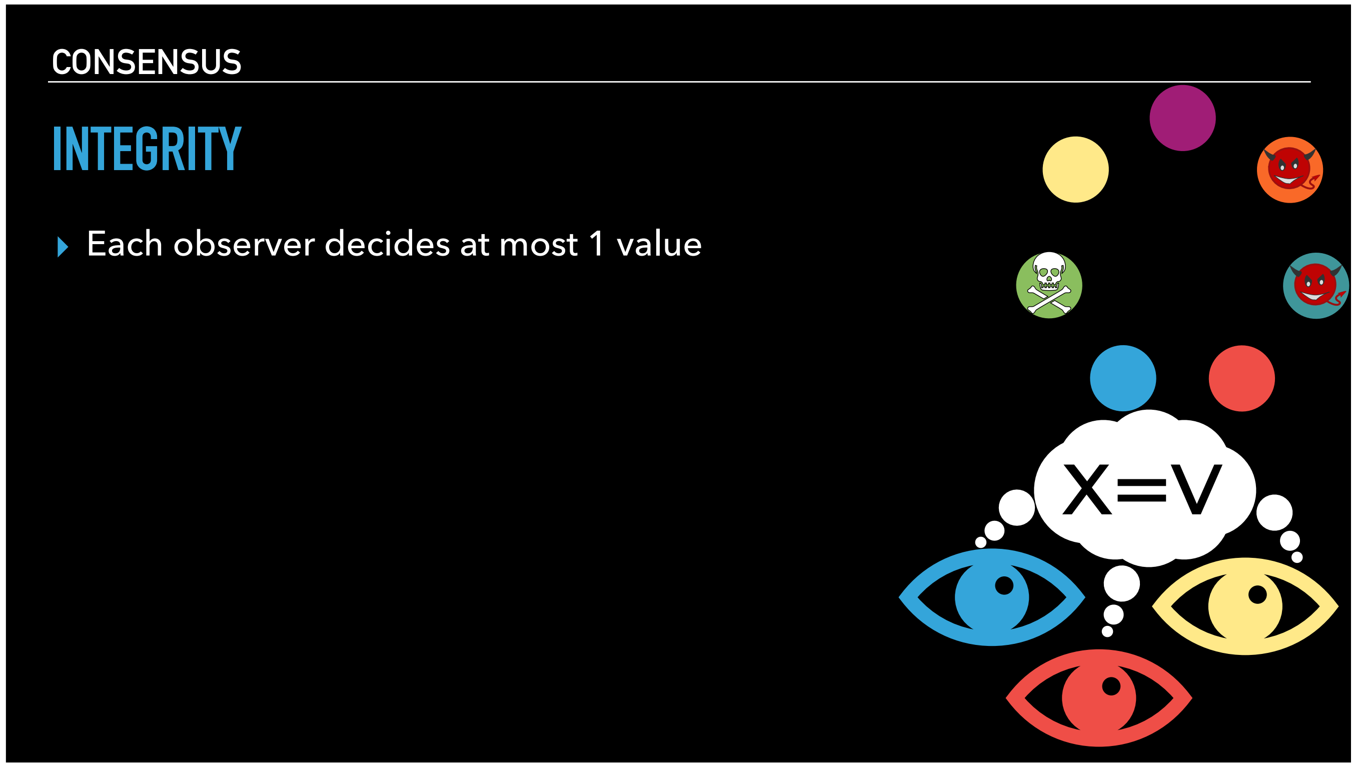
- Value decided must have been proposed by some authorized party.

  (you can't just always decide 3)



Non-triviality requires that if an observer decides something, that thing must have been proposed by some authorized party.
You can't always decide 3, because 3 may not have been proposed.

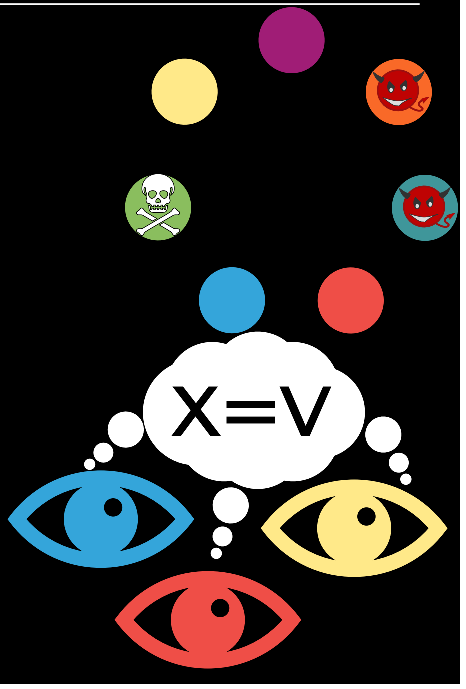
So we'll have our observers check that a value is signed by an authorized party before they decide that value.
Done.



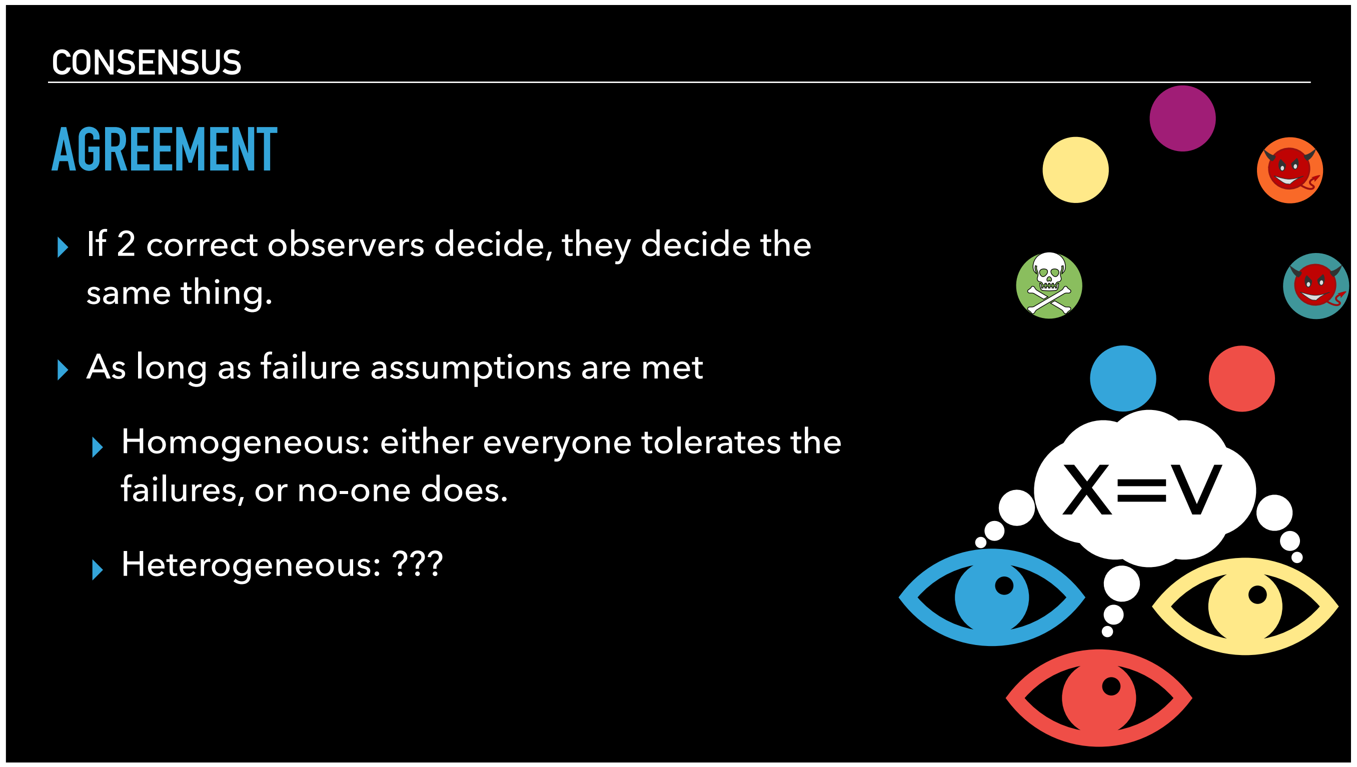
CONSENSUS

INTEGRITY

- Each observer decides at most 1 value



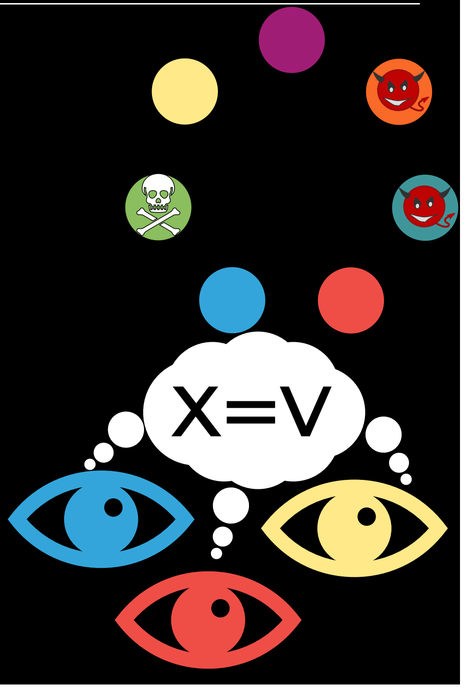
Integrity means that each observer decides at most one value.
It's not much of a consensus if you can decide two different things.
So, we'll have our observers stop once they've decided a value.
Done.



CONSENSUS

AGREEMENT

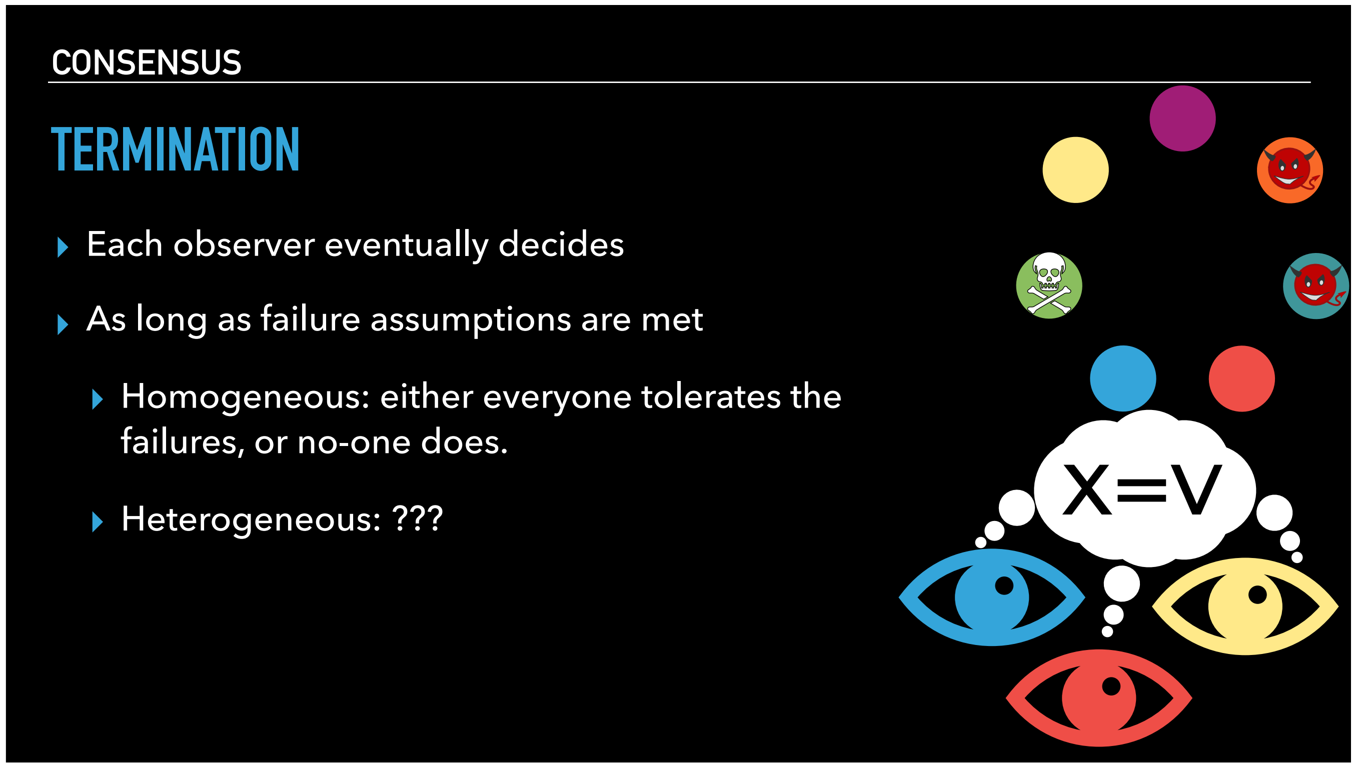
- If 2 correct observers decide, they decide the same thing.
- As long as failure assumptions are met
  - Homogeneous: either everyone tolerates the failures, or no-one does.
  - Heterogeneous: ???



Agreement is trickier.
If 2 observers decide, they decide the same thing
AS LONG AS FAILURE ASSUMPTIONS ARE MET.

In the Homogeneous case, this is clear: either there were too many failures, or there weren't.

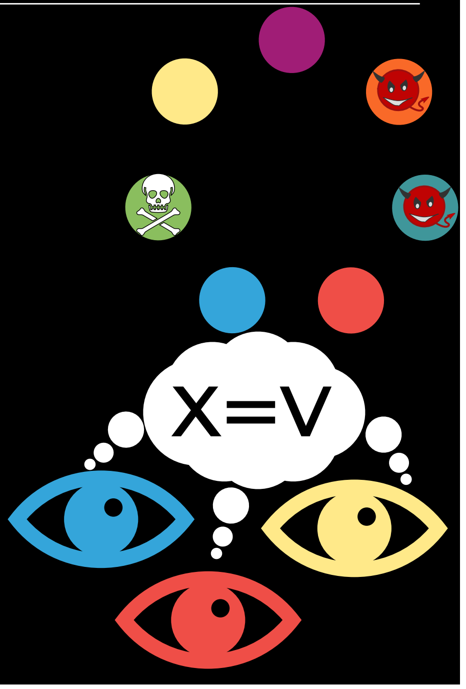
In the Heterogeneous case, not so much.
Whose failure assumptions are we talking about?
Who should agree under which assumptions?



CONSENSUS

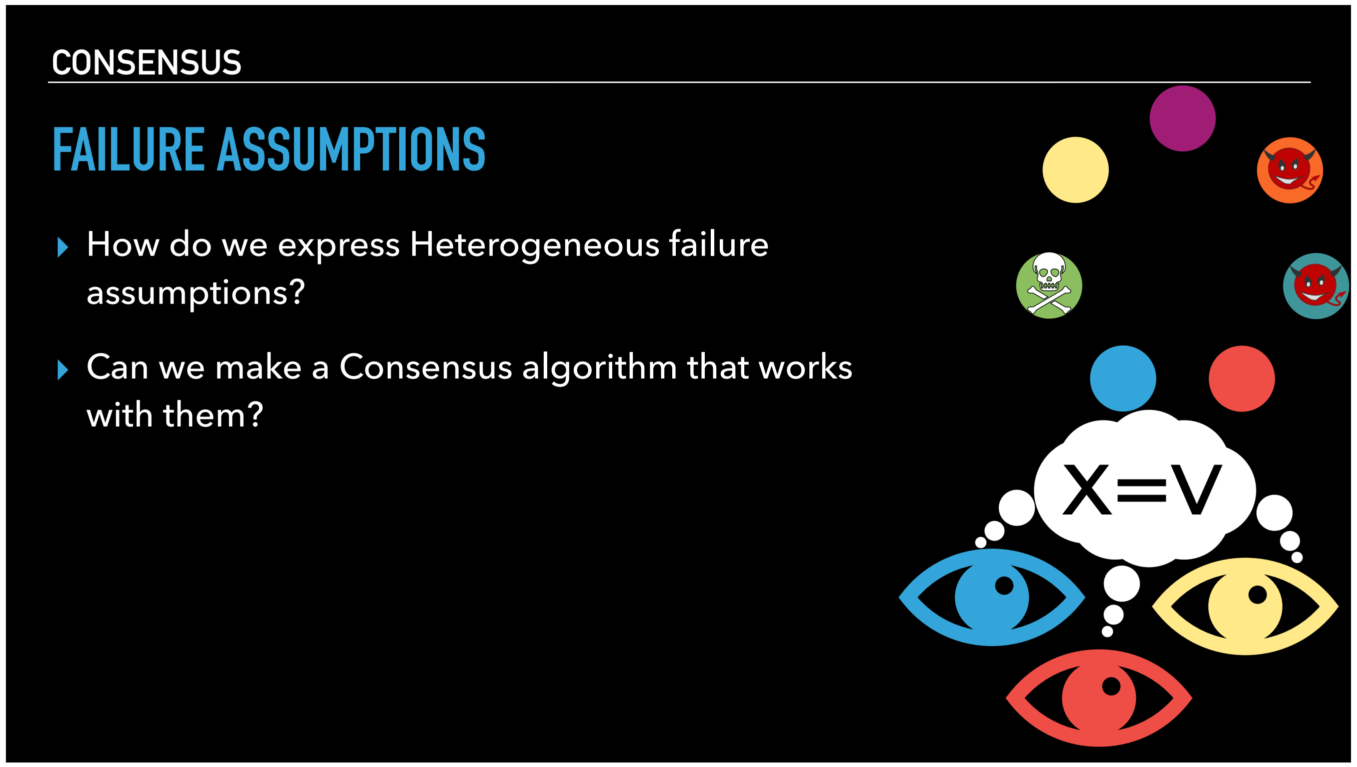
TERMINATION

- Each observer eventually decides
- As long as failure assumptions are met
  - Homogeneous: either everyone tolerates the failures, or no-one does.
  - Heterogeneous: ???



Termination's tricky too.

Obviously each observer can't decide if, say, all the participants fail.
In the homogeneous case, if there aren't too many failures, we want all the participants to decide,
but we haven't yet defined "aren't too many failures" in the heterogeneous case.



CONSENSUS

FAILURE ASSUMPTIONS

- How do we express Heterogeneous failure assumptions?
- Can we make a Consensus algorithm that works with them?

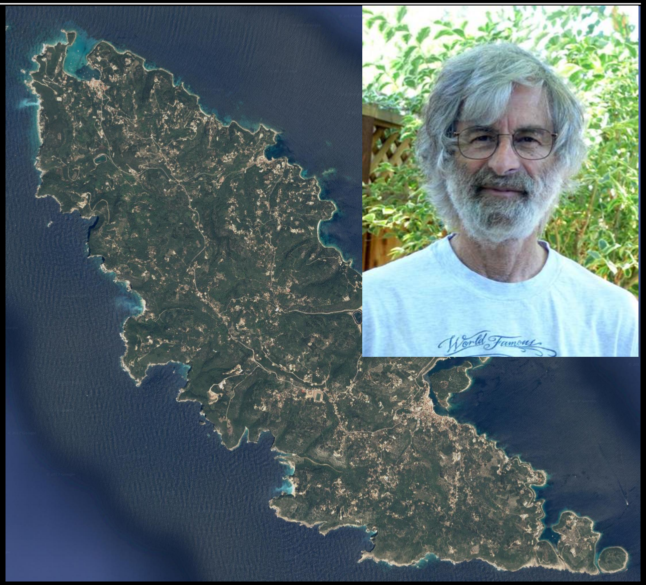
So that leaves us with 2 questions.
How do we express Heterogeneous failure assumptions, as opposed to Homogeneous ones?
And Can we make a Consensus algorithm that works under those assumptions?

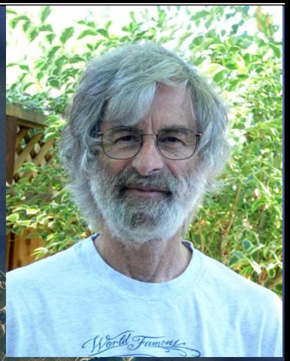
But before we can answer these, we have to …

CONSENSUS

PAXOS

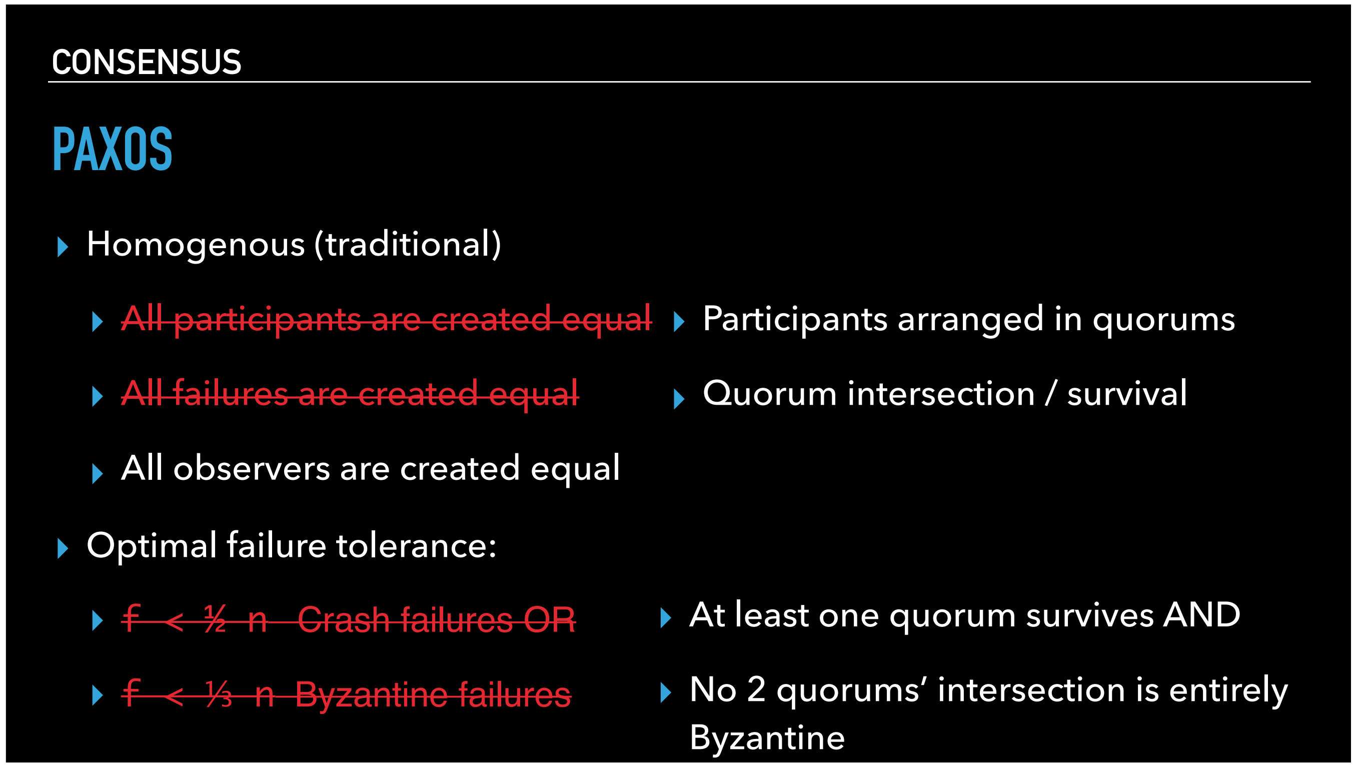
- Homogenous (traditional)
  - All participants are created equal
  - All failures are created equal
  - All observers are created equal
- Optimal failure tolerance:
  - $f < ½ n$ Crash failures OR
  - $f < ⅓ n$ Byzantine failures





… go back and learn a little more about Lamport's Paxos.

Everyone remembers Lamport's Paxos as the optimal homogenous consensus protocol.
It tolerates any f failures out of n participants, that kind of thing.
Every implementation I've come across does it that way.



CONSENSUS

PAXOS

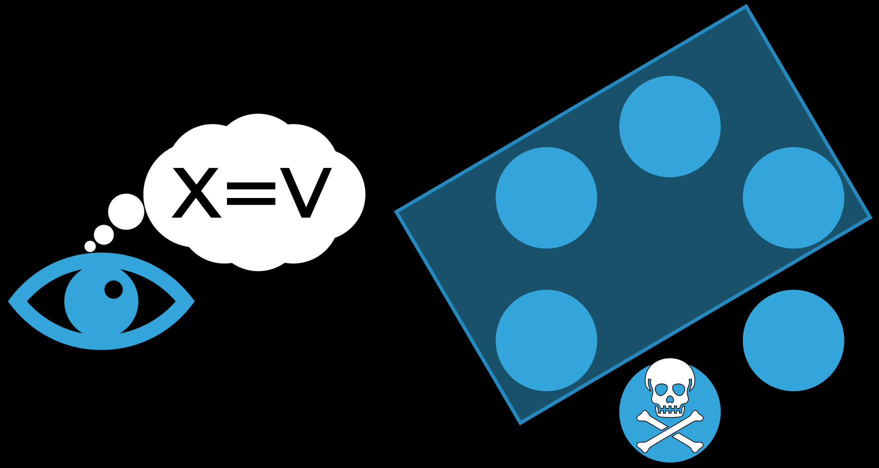
- Homogenous (traditional)
  - ~~All participants are created equal~~
  - ~~All failures are created equal~~
  - All observers are created equal
- Participants arranged in quorums
- Quorum intersection / survival
- Optimal failure tolerance:
  - ~~f < ½ n Crash failures OR~~
  - ~~f < ⅓ n Byzantine failures~~
- At least one quorum survives AND
- No 2 quorums' intersection is entirely Byzantine

But Paxos is actually defined in a much more general way, in terms of quorums, which are subsets of the participants.
When your quorums are just "any n-f participants are a quorum," it has the Homogeneous properties, but the actual requirements are more general.

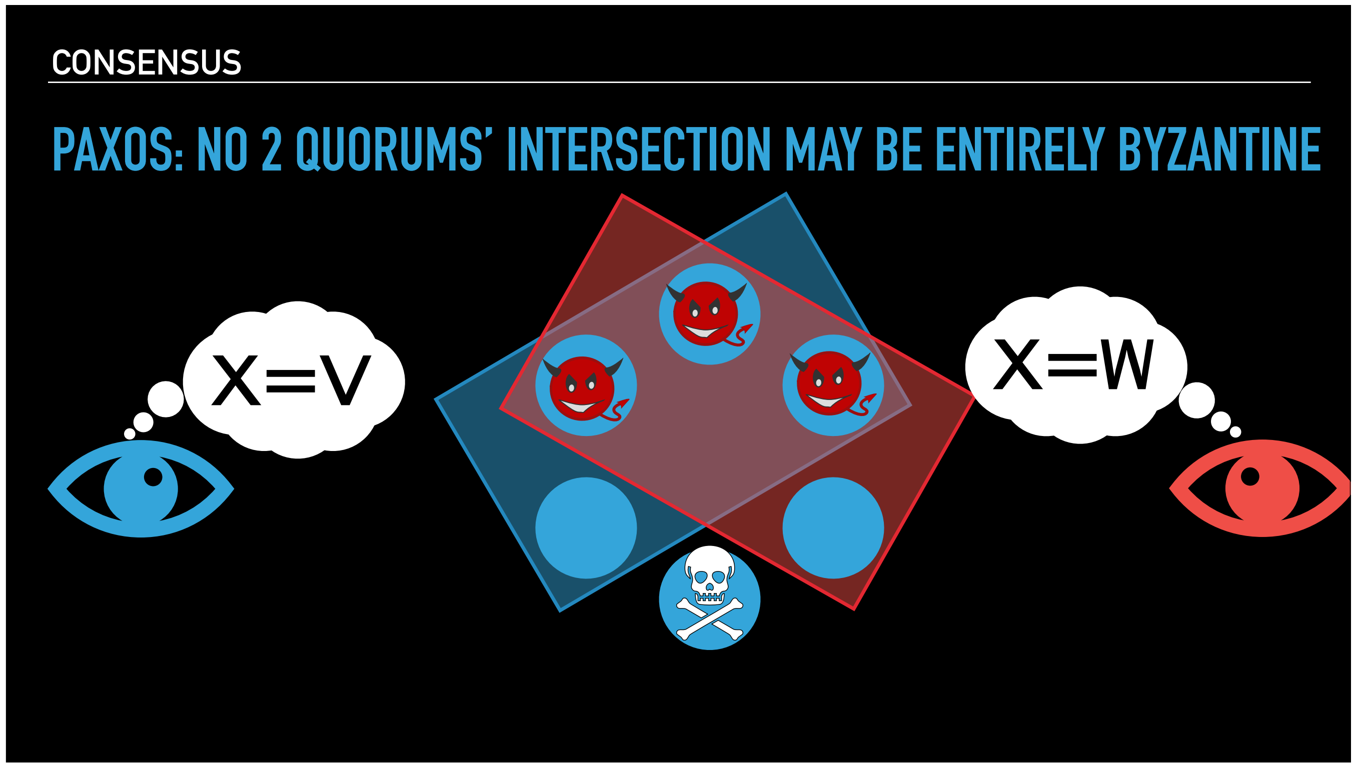
Intuitively, a quorum is a subset of participants which is sufficient to make an observer decide.
For the system to have termination, at least one quorum must survive, and
for the system to have agreement, you can't have 2 quorums with an intersection that's entirely Byzantine.

CONSENSUS

PAXOS: 1 QUORUM MUST SURVIVE



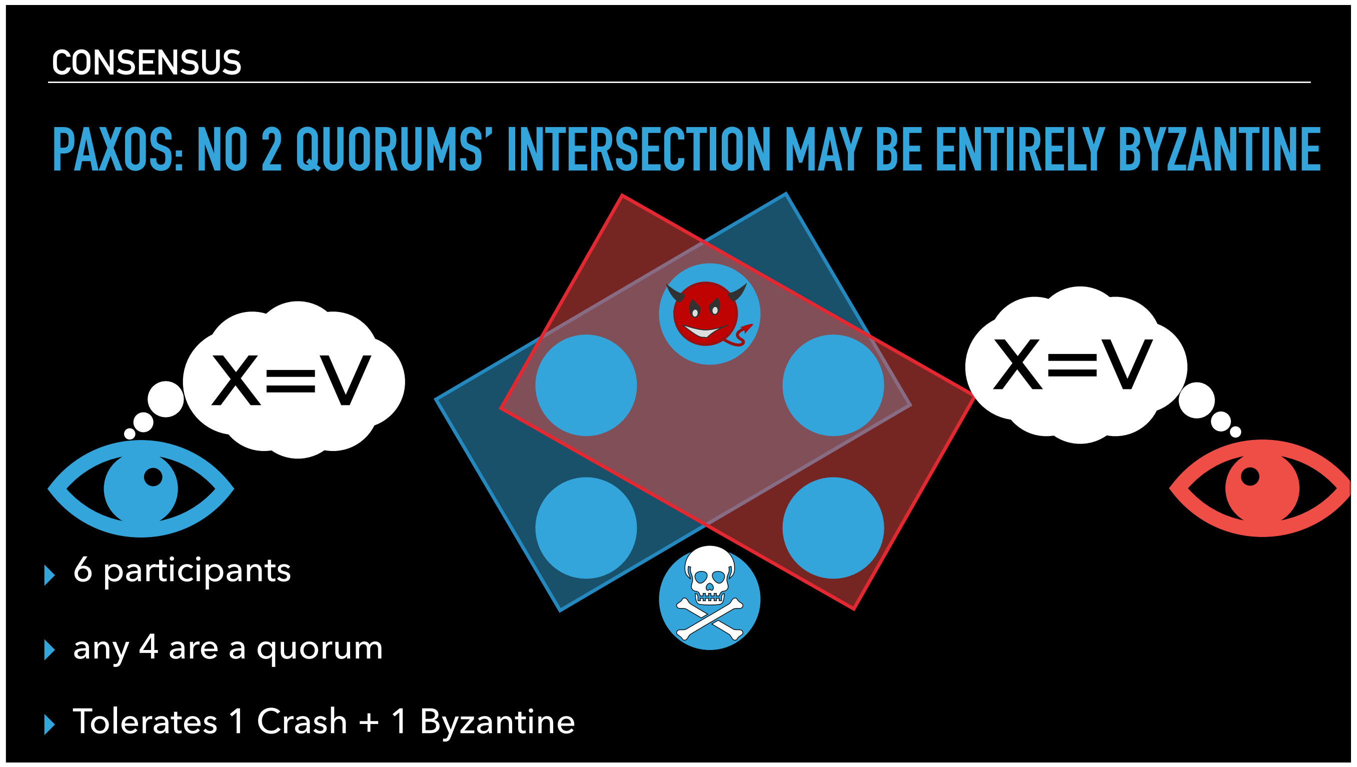
So, if there are some crash failures, but a quorum survives composed of non-crashed participants, they can allow an Observer to decide.



CONSENSUS

PAXOS: NO 2 QUORUMS' INTERSECTION MAY BE ENTIRELY BYZANTINE

But if there are two quorums whose intersection is wholly Byzantine, then
Each byzantine participant can “bi-simulate,” meaning act one way for the purposes of one observer, and act another way for the purposes of another observer.
That means that each quorum can make a participant decide, and they don’t necessarily make them decide the same thing.
We don’t have agreement.



CONSENSUS

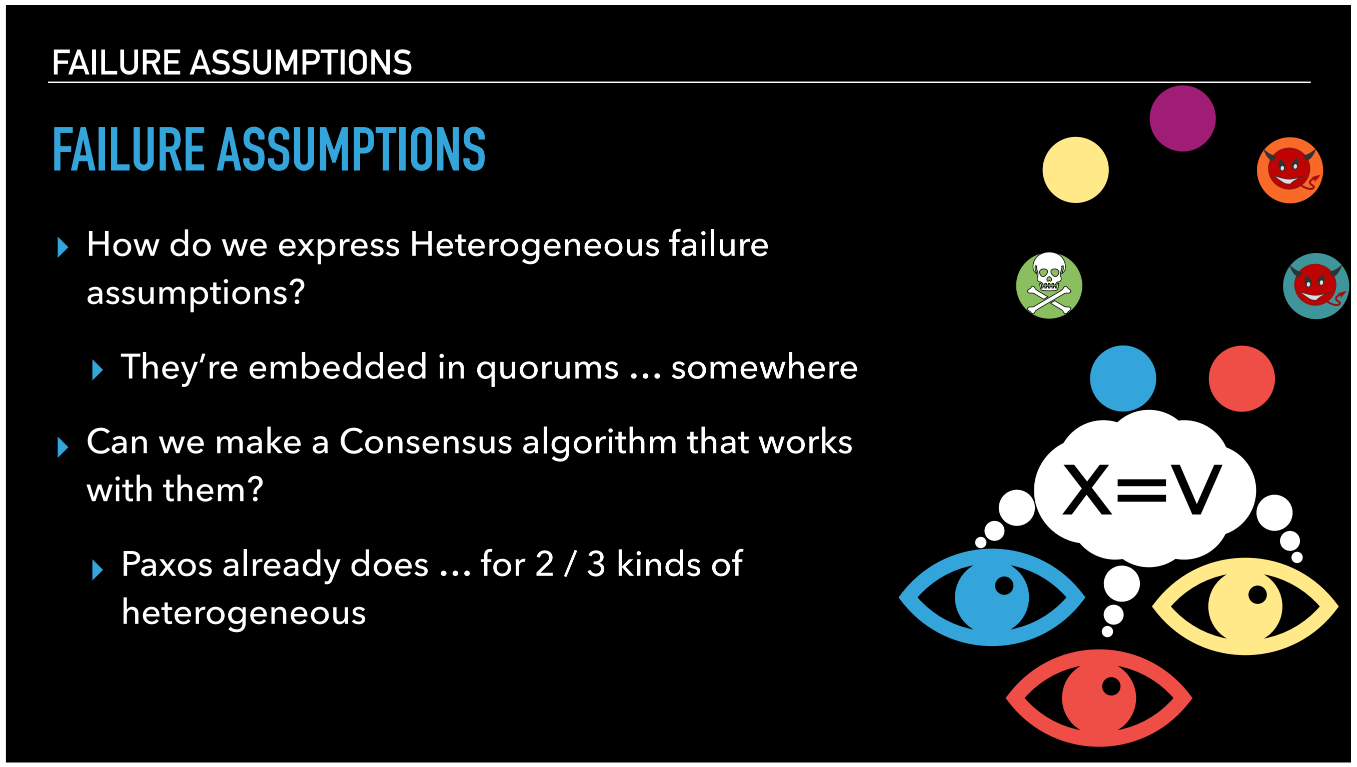
PAXOS: NO 2 QUORUMS' INTERSECTION MAY BE ENTIRELY BYZANTINE

- 6 participants
- any 4 are a quorum
- Tolerates 1 Crash + 1 Byzantine

If there are safe participants in the quorum intersection, than that can't happen: the safe participant won't bisimulate.

Now, it so happens that with 6 participants, you can make a Paxos where any 4 are a quorum, and this Paxos will tolerate 1 byzantine and an additional crash failure, but not two Byzantine failures.

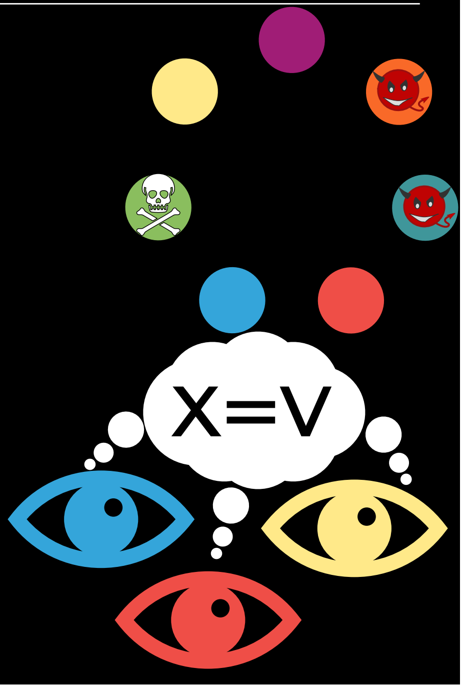
So this quorum-based definition is already at least a little bit Heterogeneous.



FAILURE ASSUMPTIONS

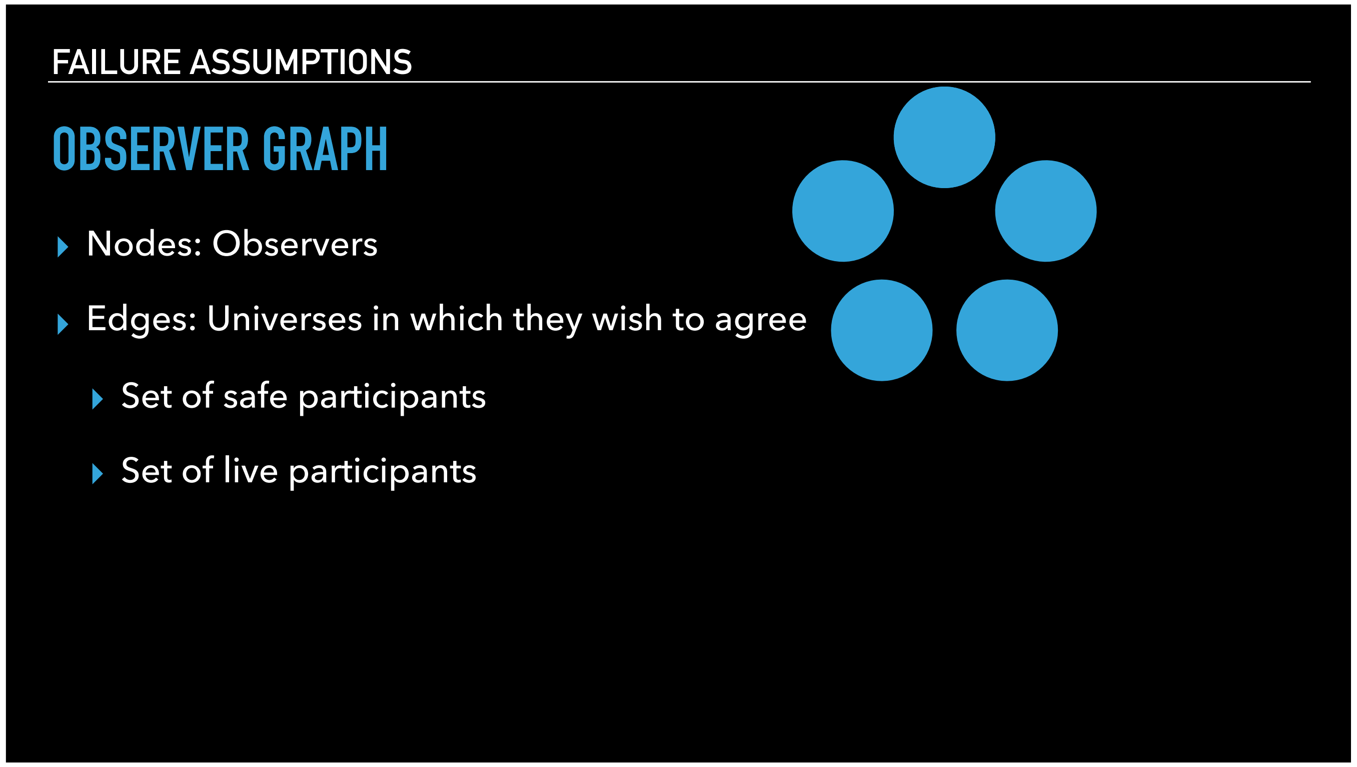
FAILURE ASSUMPTIONS

- How do we express Heterogeneous failure assumptions?
  - They're embedded in quorums … somewhere
- Can we make a Consensus algorithm that works with them?
  - Paxos already does … for 2 / 3 kinds of heterogeneous



So, returning to our questions,
How do we express Heterogeneous Failure assumptions?
  - Well, with quorum-based Paxos, they're embedded somewhere in the definitions of those quorums.
Can we make a Consensus algorithm that works?
  - Well, Paxos already is, for 2 out of the 3 kinds of Heterogeneity we've listed.
Specifically, not all participants are created equal, and not all failures are created equal.

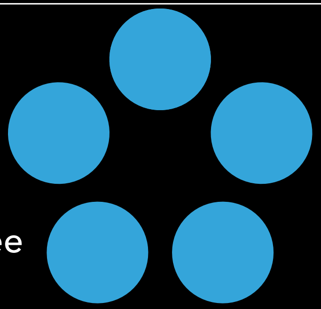
But what we really *want* is to be able to express our Observer's assumptions in advance, and then see if we can get a consensus protocol from them.



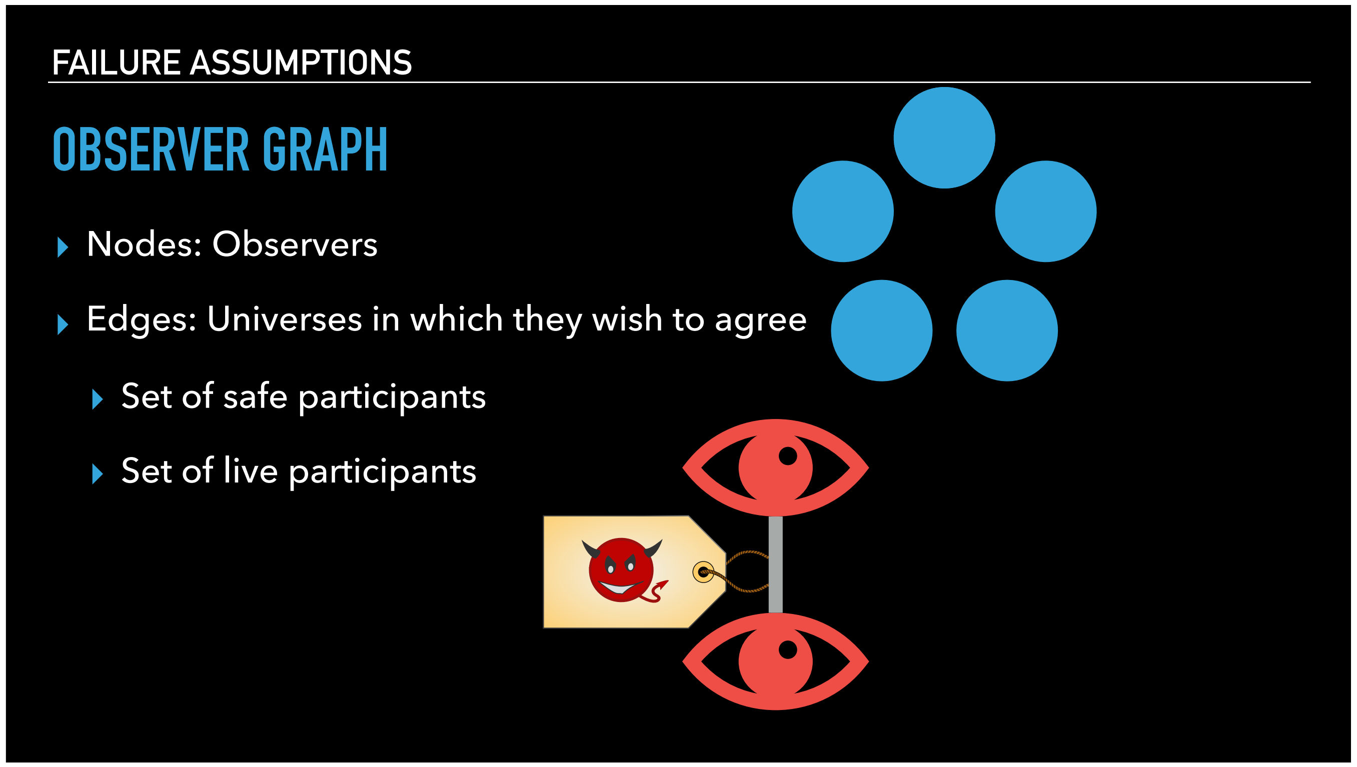
FAILURE ASSUMPTIONS

OBSERVER GRAPH

- Nodes: Observers
- Edges: Universes in which they wish to agree
    - Set of safe participants
    - Set of live participants



And we're going to do it using a thing we call an Observer Graph.
On the observer graph, the nodes are observers, and each edge is labeled with, in a mathematical sense,
the set of all Universes in which that pair of observers wish to agree.
Specifically, a "universe" means:
- The set of participants who are "safe" in that universe: they're not Byzantine, paired with:
- The set of participants who are "live" in that universe: they're not crashed.



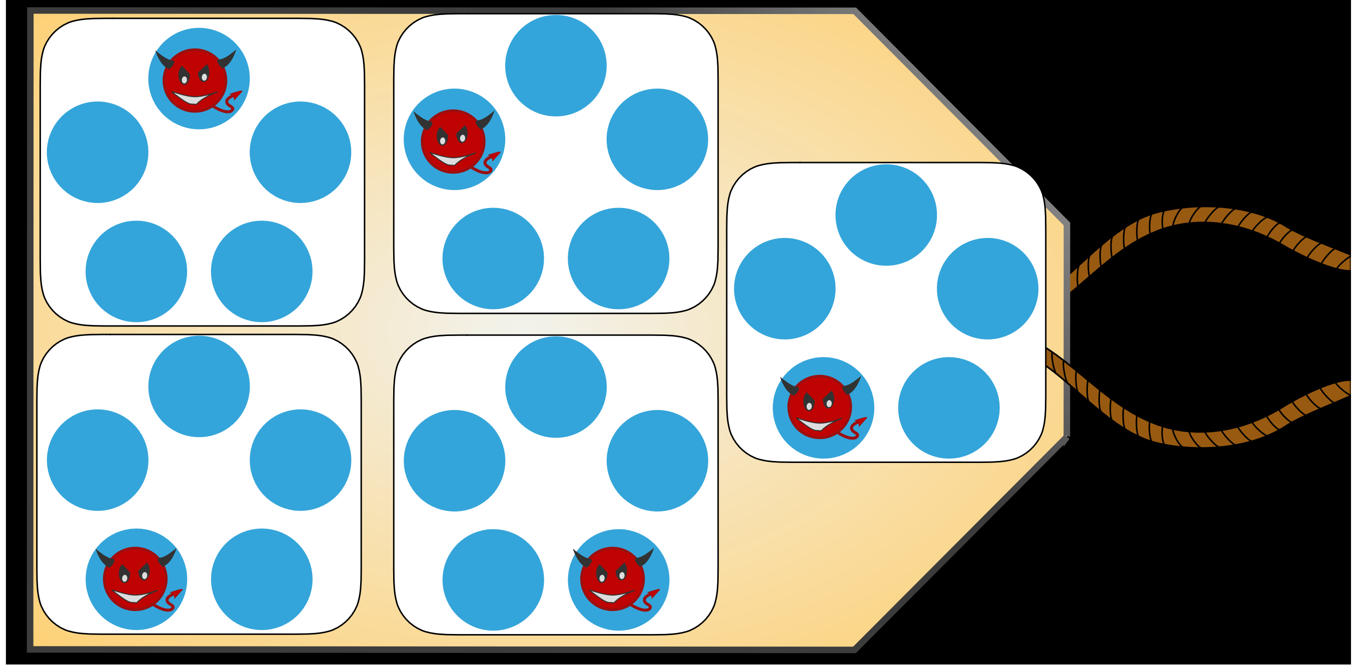
FAILURE ASSUMPTIONS

OBSERVER GRAPH

- Nodes: Observers
- Edges: Universes in which they wish to agree
  - Set of safe participants
  - Set of live participants

So suppose we have a pair of observers, connected by an edge.
And that edge has a label representing that these observers want to agree even when there’s one byzantine failure.

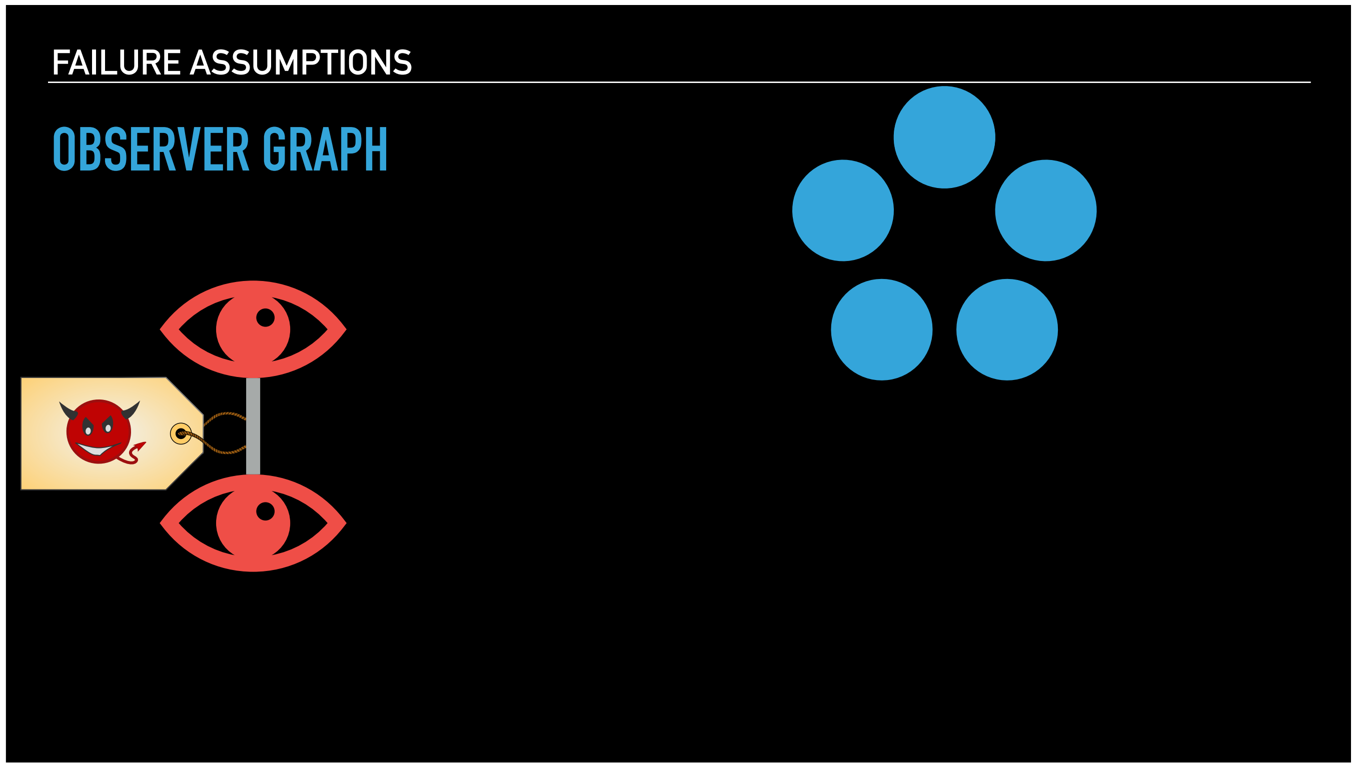
FAILURE ASSUMPTIONS



So, technically, that label has all the possible universes with a byzantine failure in it.

Actually, since a Byzantine node can act like a crashed node or a correct node, all possible single crashes and the no-failure universes should be on here too, but I ran out of space to draw them.

Anyway,



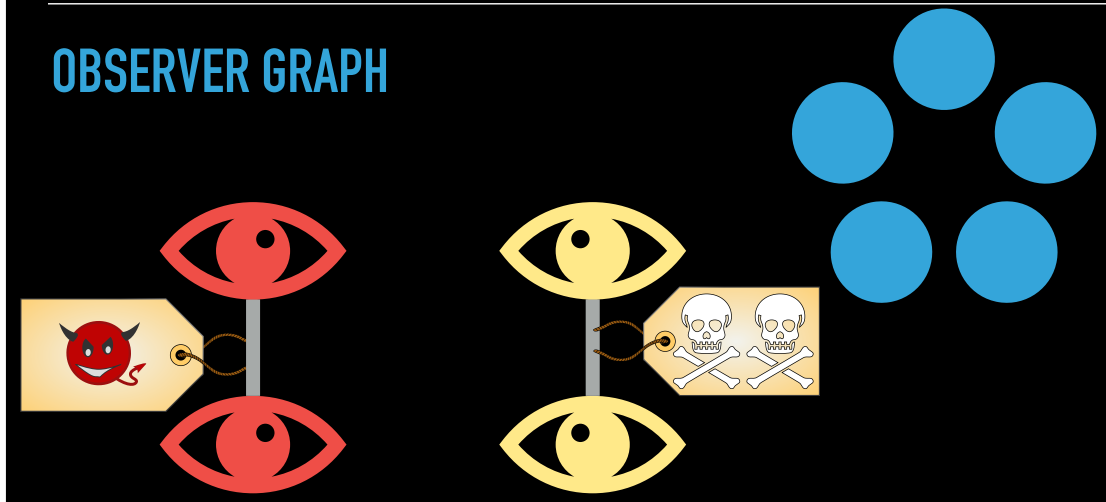
FAILURE ASSUMPTIONS

OBSERVER GRAPH

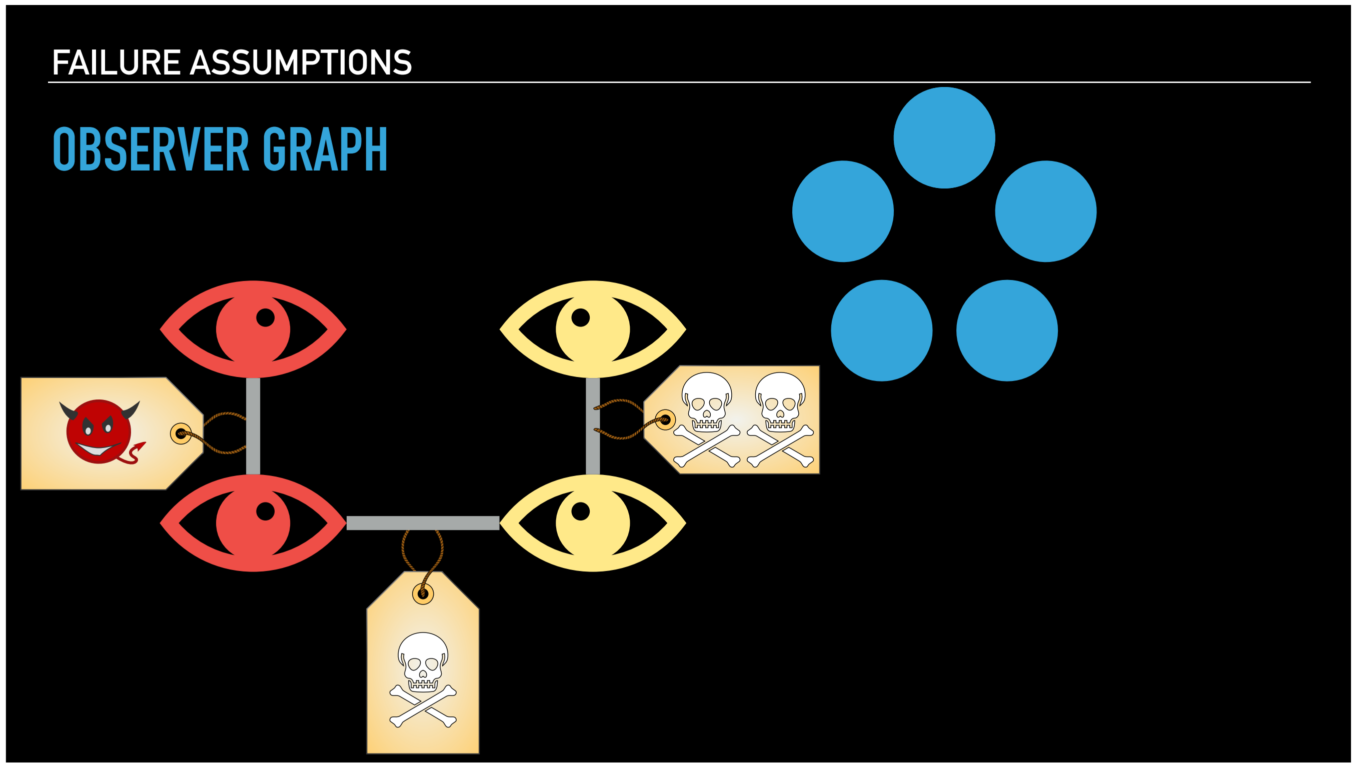
The two red observers want to agree when there’s a byzantine failure.

FAILURE ASSUMPTIONS

OBSERVER GRAPH

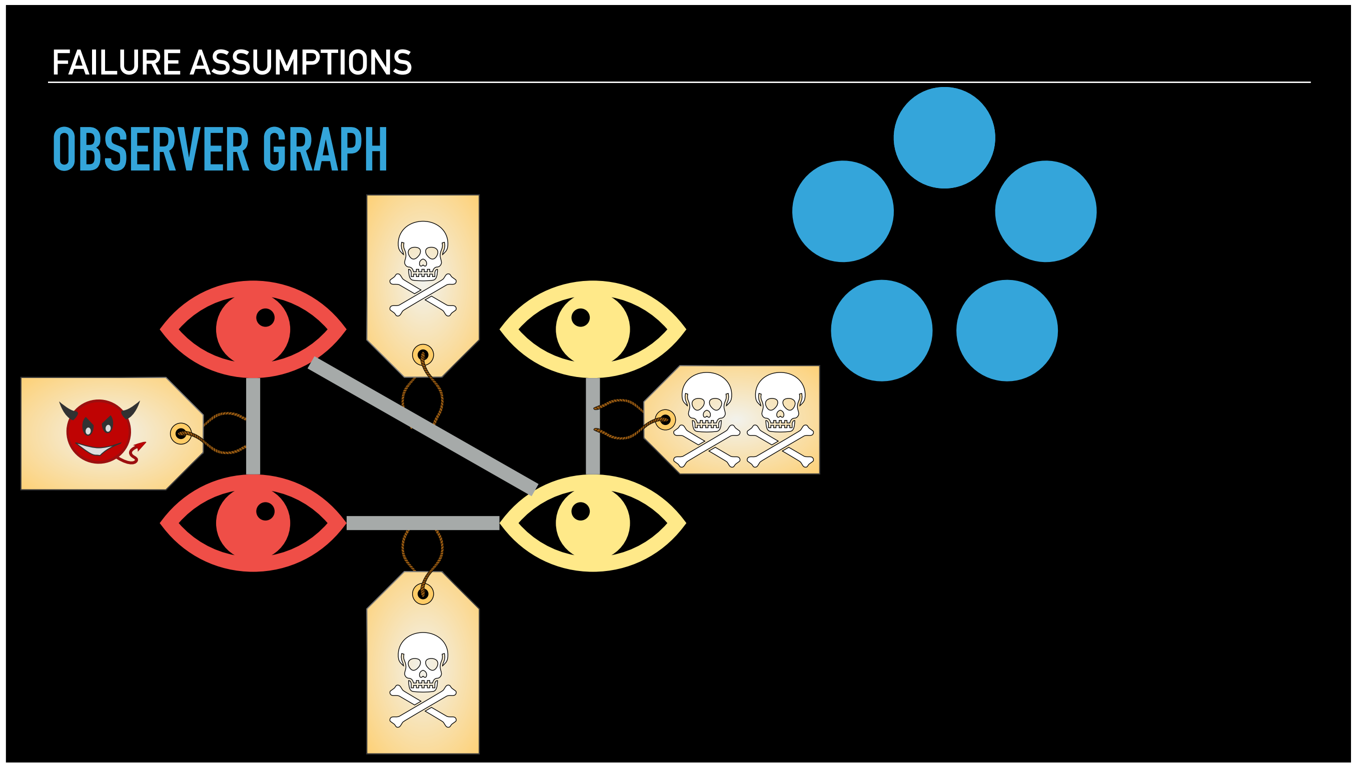


Suppose there are also two yellow observers who want to agree even if two participants *crash*, but aren’t worried about byzantine failures.



And maybe the bottom two observers like the idea of agreeing too, but it’s not as important that *they* agree: if there’s anything more than one crash failure, they don’t care.

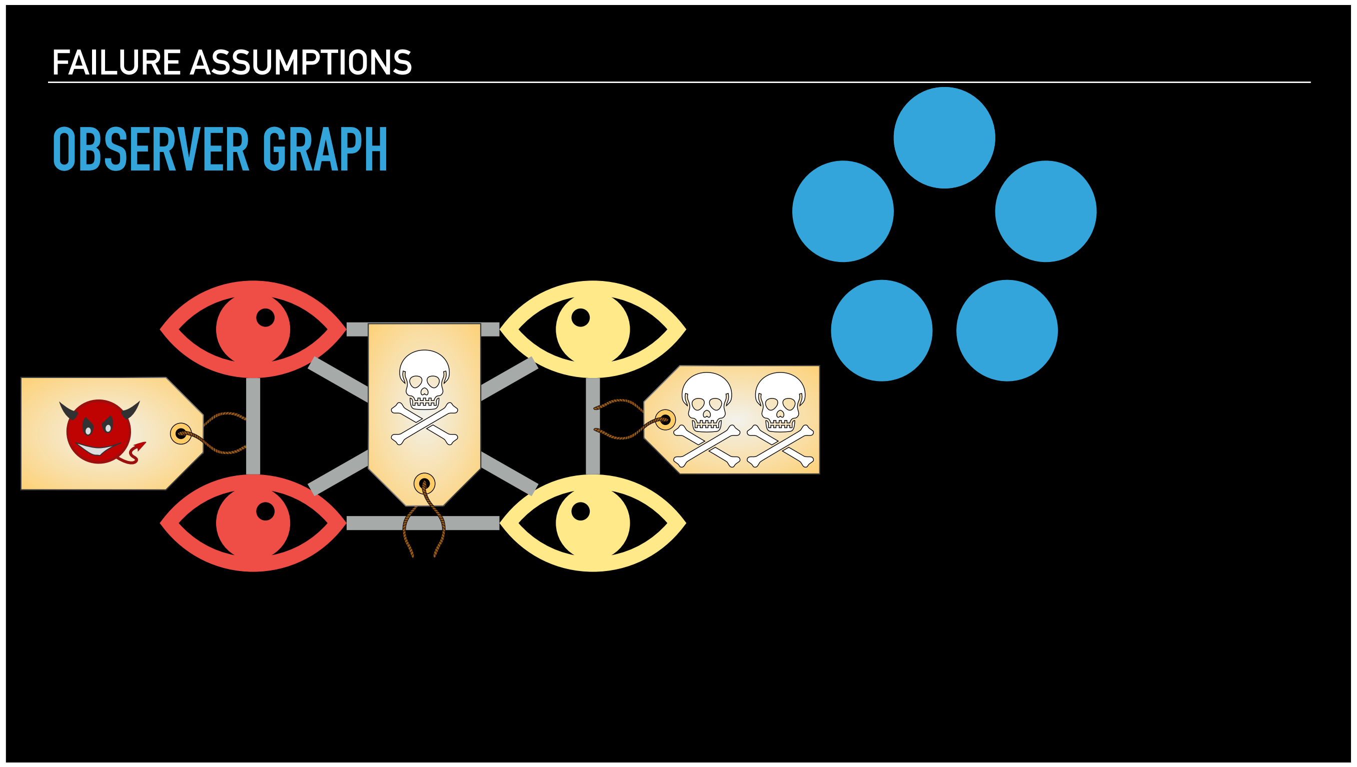
But, agreement is transitive.
So if there is at most one crash failure, the two red observers will agree, and the two bottom observers will agree, which means that



FAILURE ASSUMPTIONS

OBSERVER GRAPH

The top left Red and the bottom Right yellow will agree exactly when there’s one crash failure.


FAILURE ASSUMPTIONS

OBSERVER GRAPH

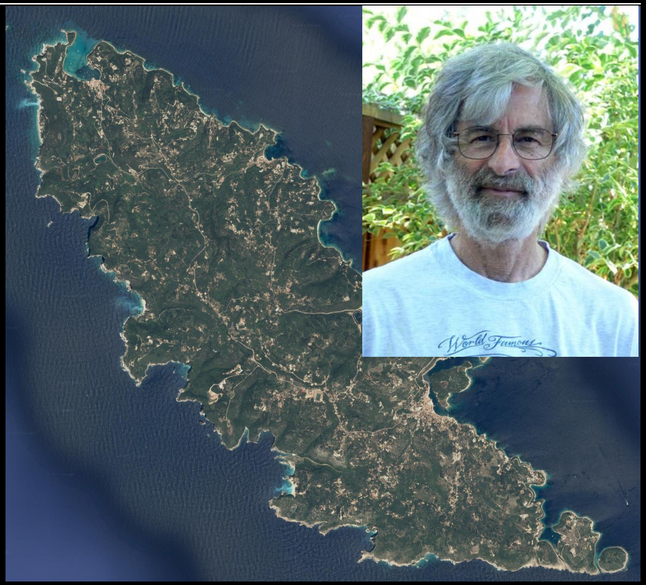
In fact similar rules hold for all these Red-Yellow edges, all of which have single-crash-failure labels.

For what it’s worth, we do have a mathematical formulation for how to apply all of these transitivity rules and get a “condensed observer graph.”

Now, it so happens that there is a consensus algorithm that satisfies this observer graph with these five participants, but to understand it, we need to …

PAXOS

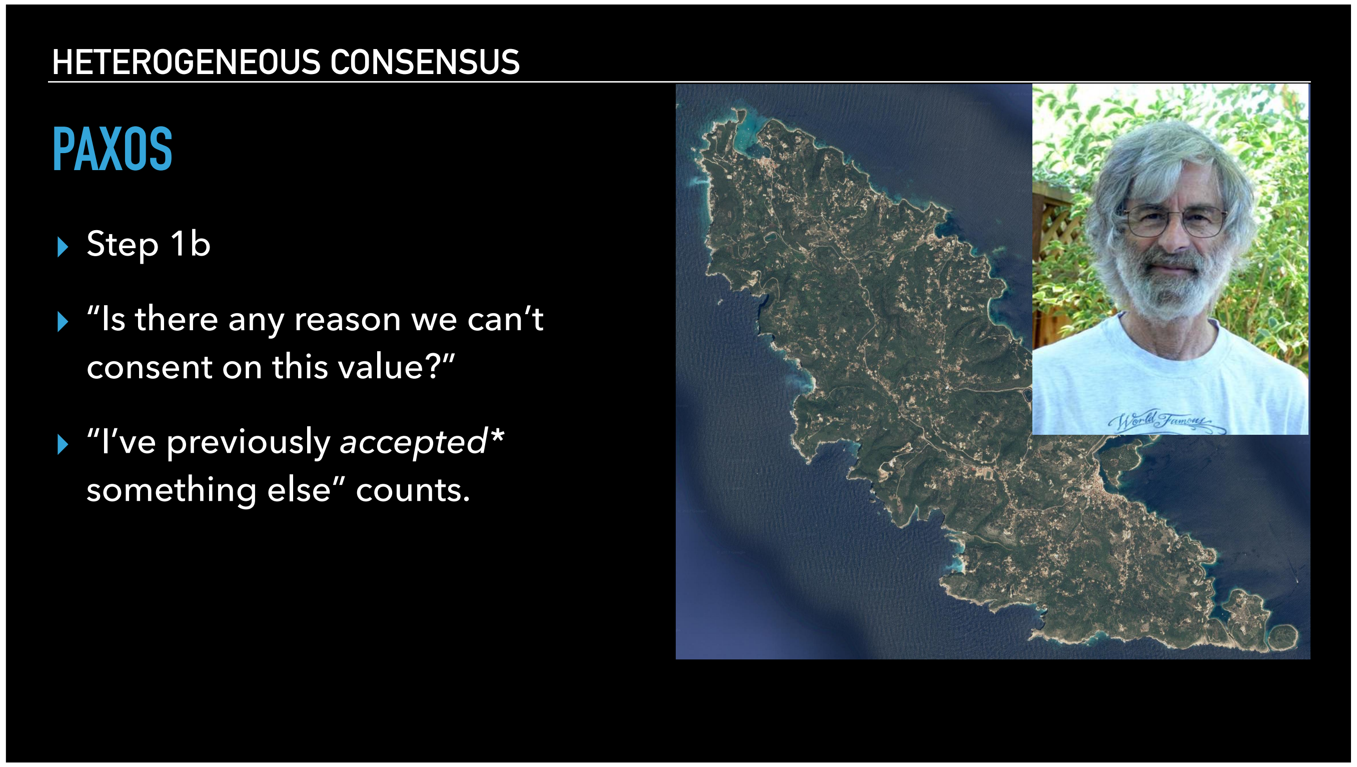
- Step 1b



… take one more look at paxos.

Paxos involves a step called “1b,”

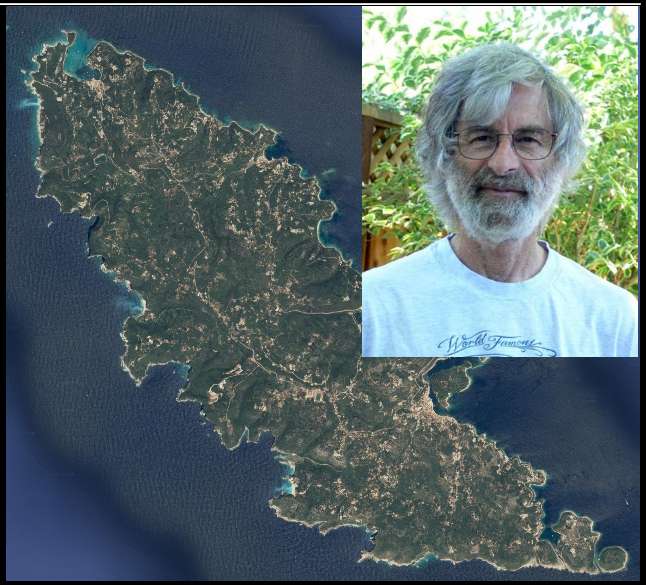
CLICK



HETEROGENEOUS CONSENSUS

PAXOS

- Step 1b
- “Is there any reason we can’t consent on this value?”
- “I’ve previously *accepted** something else” counts.

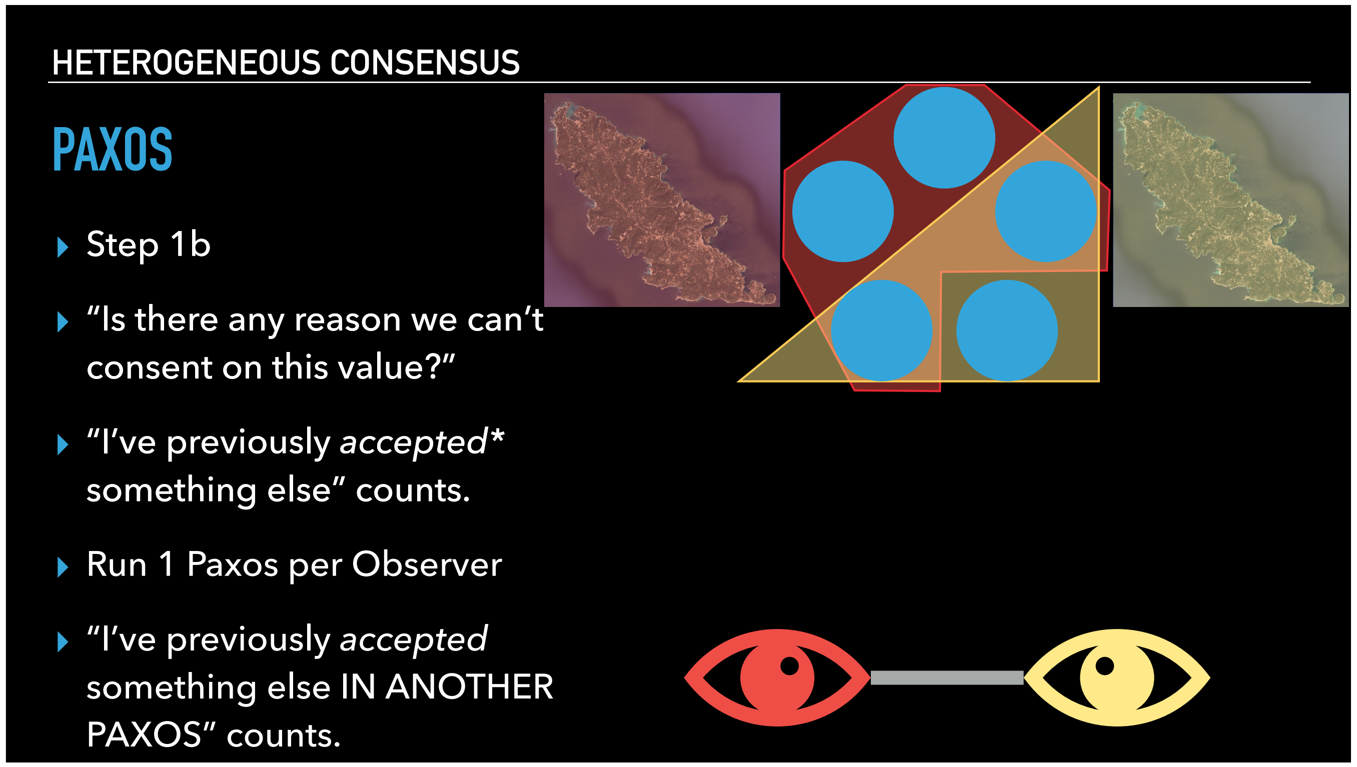


In which participants respond to the question “Is there any reason we can’t consent on this value?”

And the reason “I’ve previously accepted something else” counts.

Now I should note that “accept” is a thing participants do with values in Paxos, and it’s not the same as when observers “decide,”
However, this is the step that ensures intersecting quorums can’t decide different things.
The participants in the intersection will have a reason to put in their 1bs.

So, the principal change we’re going to make to Byzantine Paxos is:
CLICK



HETEROGENEOUS CONSENSUS

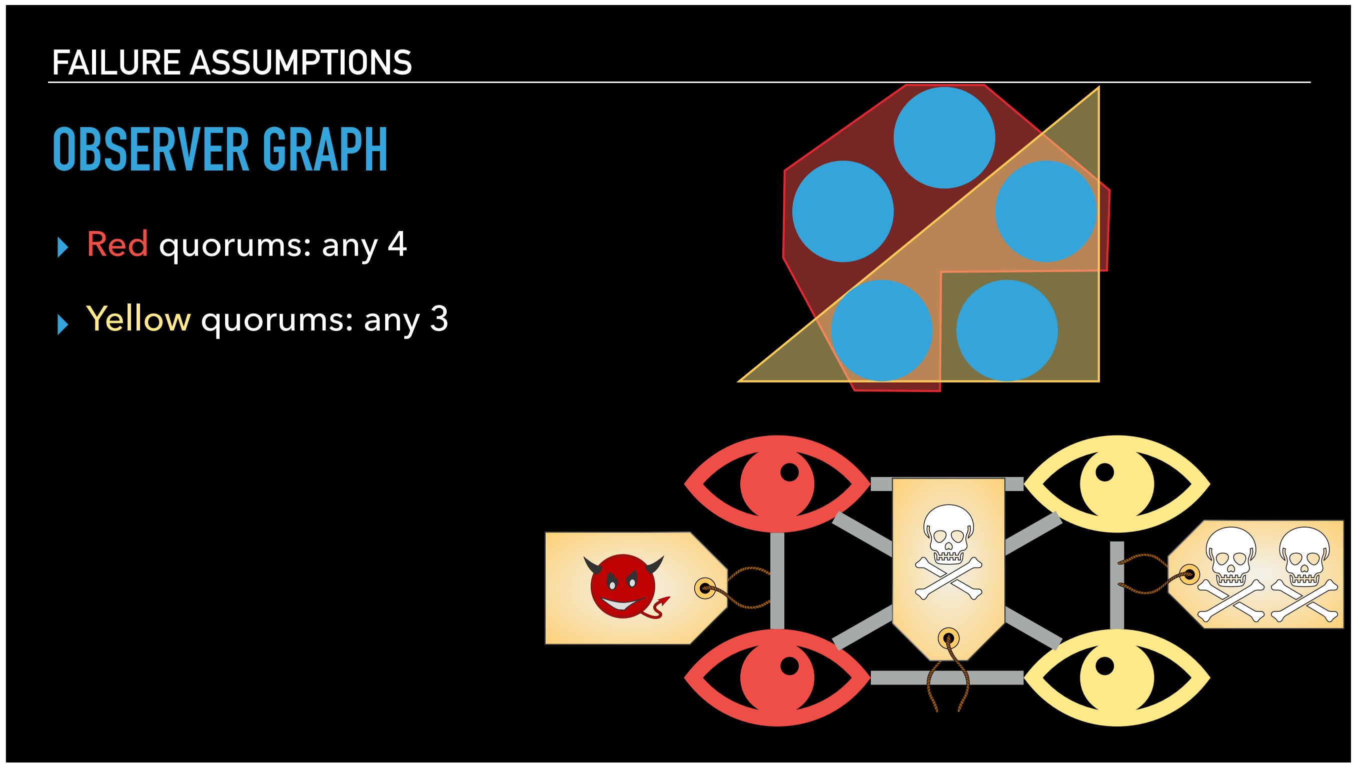
PAXOS

- Step 1b
- “Is there any reason we can’t consent on this value?”
- “I’ve previously *accepted** something else” counts.
- Run 1 Paxos per Observer
- “I’ve previously *accepted* something else IN ANOTHER PAXOS” counts.

We run a different Paxos for each observer,
so if we’re talking about these two observers, we run two different paxoses, one red and one yellow,

And in these Paxoses, “I’ve previously accepted something else IN ANOTHER PAXOS” counts as a reason in step 1b.
That prevents the two observers from deciding different things, under some conditions.

Now, I know what you’re thinking:
“Isaac, that sounds terribly inefficient!”
Well, in reality, the vast majority of conceptual messages for each *conceptual* Paxos can share a physical instantiation with a message from other Paxoses.
Most notably, if all the observers’ connections have the same label, we’re exactly the same as regular Paxos.



FAILURE ASSUMPTIONS

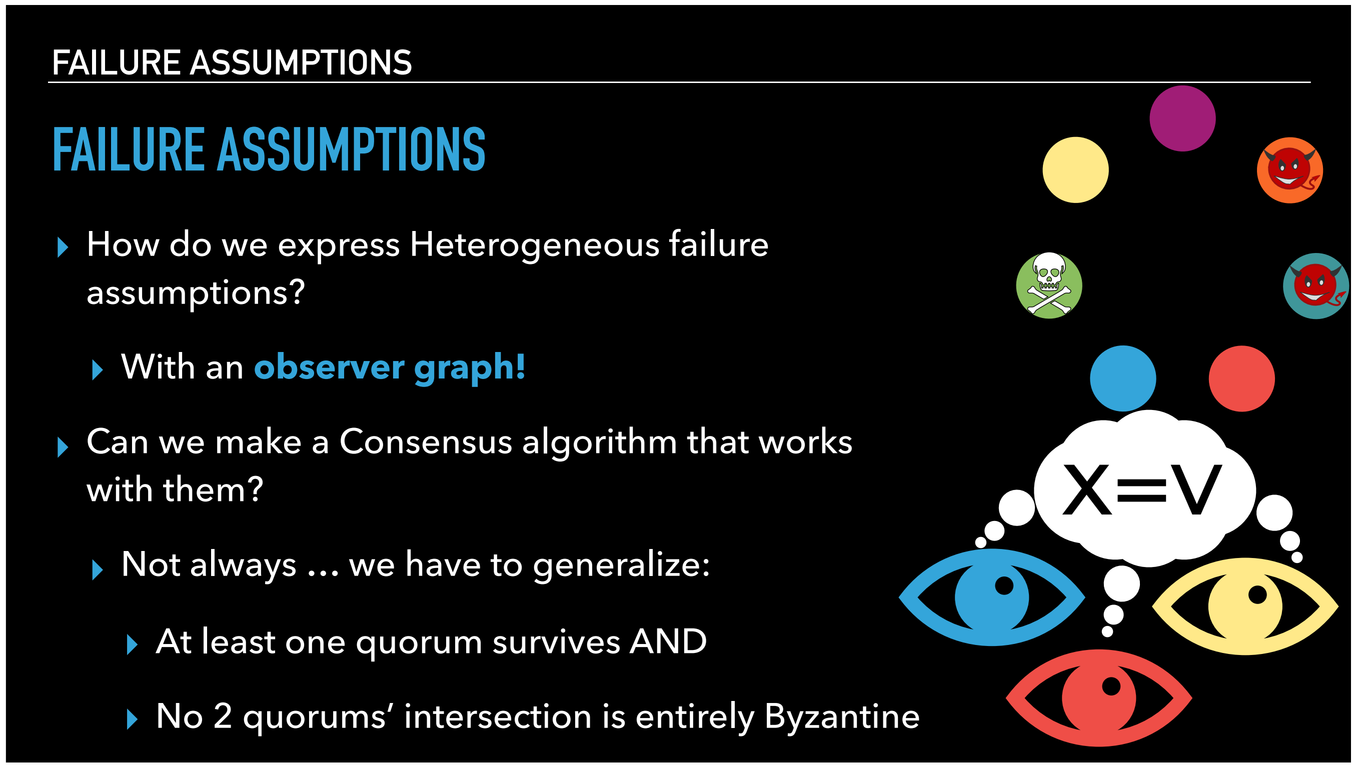
OBSERVER GRAPH

- Red quorums: any 4
- Yellow quorums: any 3

Now for this graph, the Paxoses for the Red observers have quorums of any 4 participants,
and for the yellow participants any 3.

So long as there is at most one crash failure, There will be one live yellow and red quorum, and they all intersect.
If there is a Byzantine Failure, the Red observers will agree, but the Yellow observers may not.
And if there are two crashes, the yellow observers will agree and terminate, but the Red observers will never terminate.

So at last we have a consensus with the third kind of Heterogeneity: not all observers are created equal.


FAILURE ASSUMPTIONS

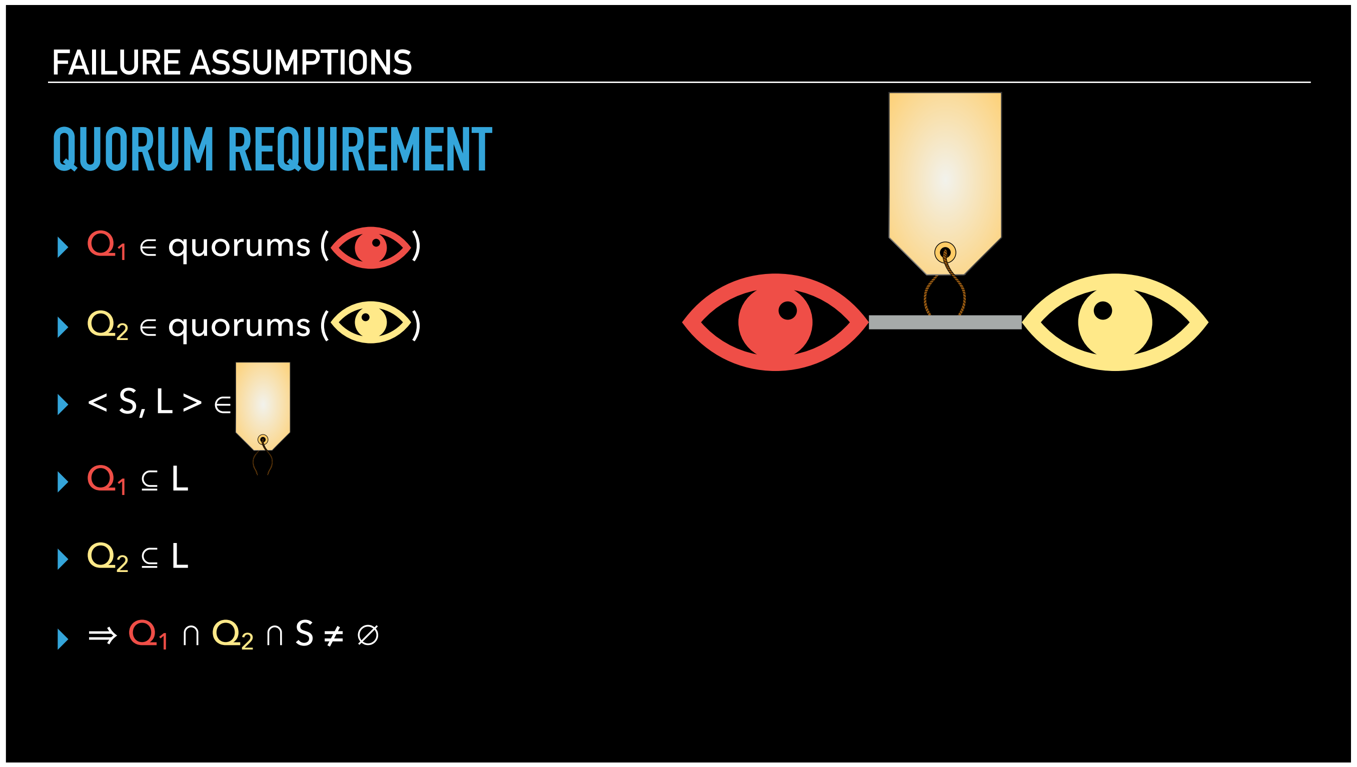
FAILURE ASSUMPTIONS

- How do we express Heterogeneous failure assumptions?
  - With an **observer graph!**
- Can we make a Consensus algorithm that works with them?
  - Not always … we have to generalize:
    - At least one quorum survives AND
    - No 2 quorums' intersection is entirely Byzantine

But returning to our questions,

How do we express Heterogeneous failure assumptions?
- With an observer graph:
  Each pair of observers express the possible universes in which they wish to agree.

Can we make a Consensus algorithm that works with them?
- Well clearly sometimes yes, because we just did one.
  But also sometimes no. If two observers want to agree even when all participants fail, that's clearly not possible.
  We need to generalize Paxos' Quorum requirements.
  Recall that Paxos requires that one quorum survives, and that no 2 quorums' intersection is entirely Byzantine.



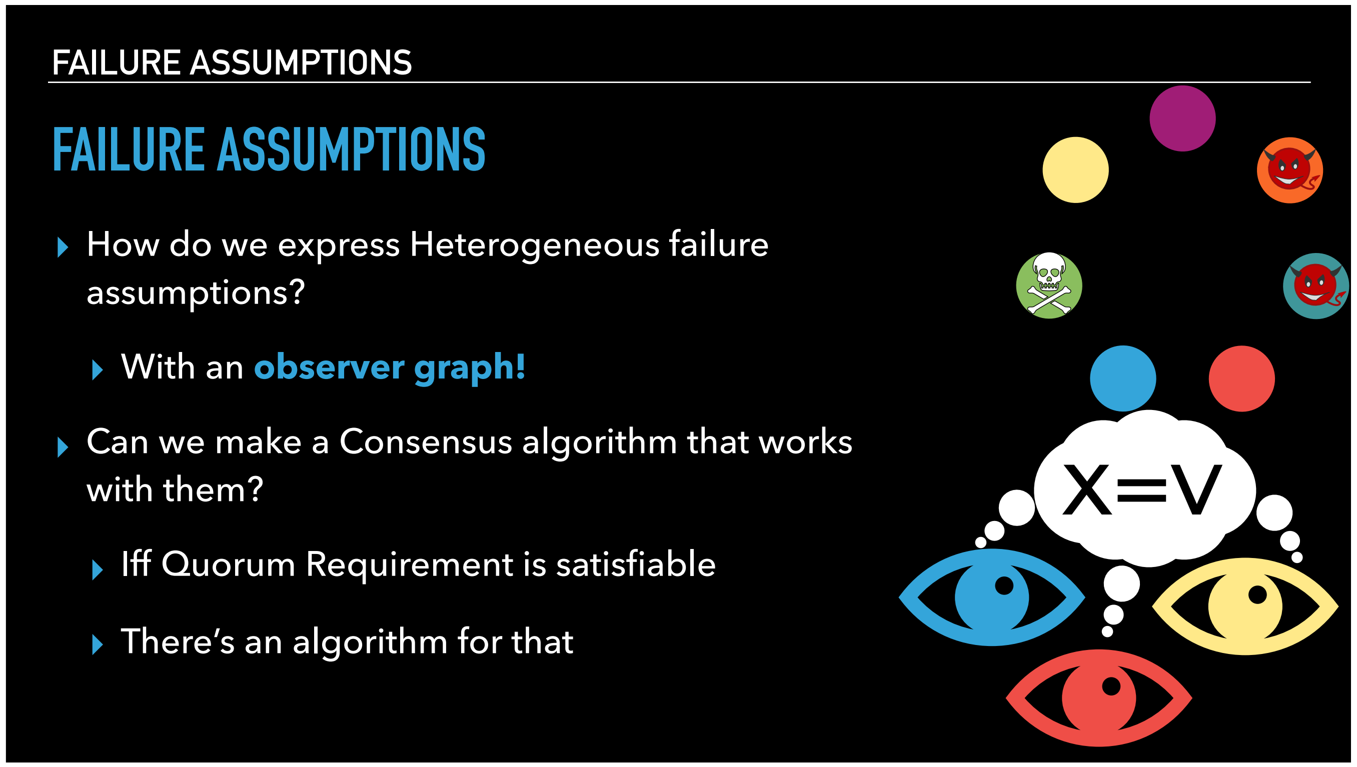
FAILURE ASSUMPTIONS

QUORUM REQUIREMENT

- $Q_1 \in$ quorums (👁)
- $Q_2 \in$ quorums (👁)
- $< S, L > \in$ 🏷
- $Q_1 \subseteq L$
- $Q_2 \subseteq L$
- $\Rightarrow Q_1 \cap Q_2 \cap S \neq \varnothing$

And this is what our generalization looks like.
If we have these two observers, Red and Yellow,
and they want to agree under conditions in this label,
Then for each quorum Q1 for Red, and Q2 for Yellow, and
for each Universe, meaning set S of safe participants, and set L of live participants in the Label,
if both quorums are live, meaning they're both subsets of the live participants,
then the intersection of the quorums must have at least one safe participant: it can't have an empty intersection with S.
This is our version of the no Byzantine intersection rule.

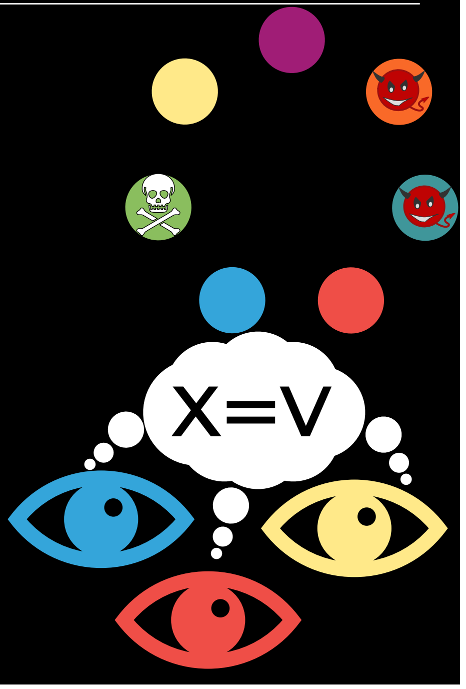
Furthermore, for all universes in that label, there must exist a quorum of Red, and a quorum of Yellow that meet this requirement.
This is our version of the "a quorum survives" rule.



FAILURE ASSUMPTIONS

FAILURE ASSUMPTIONS

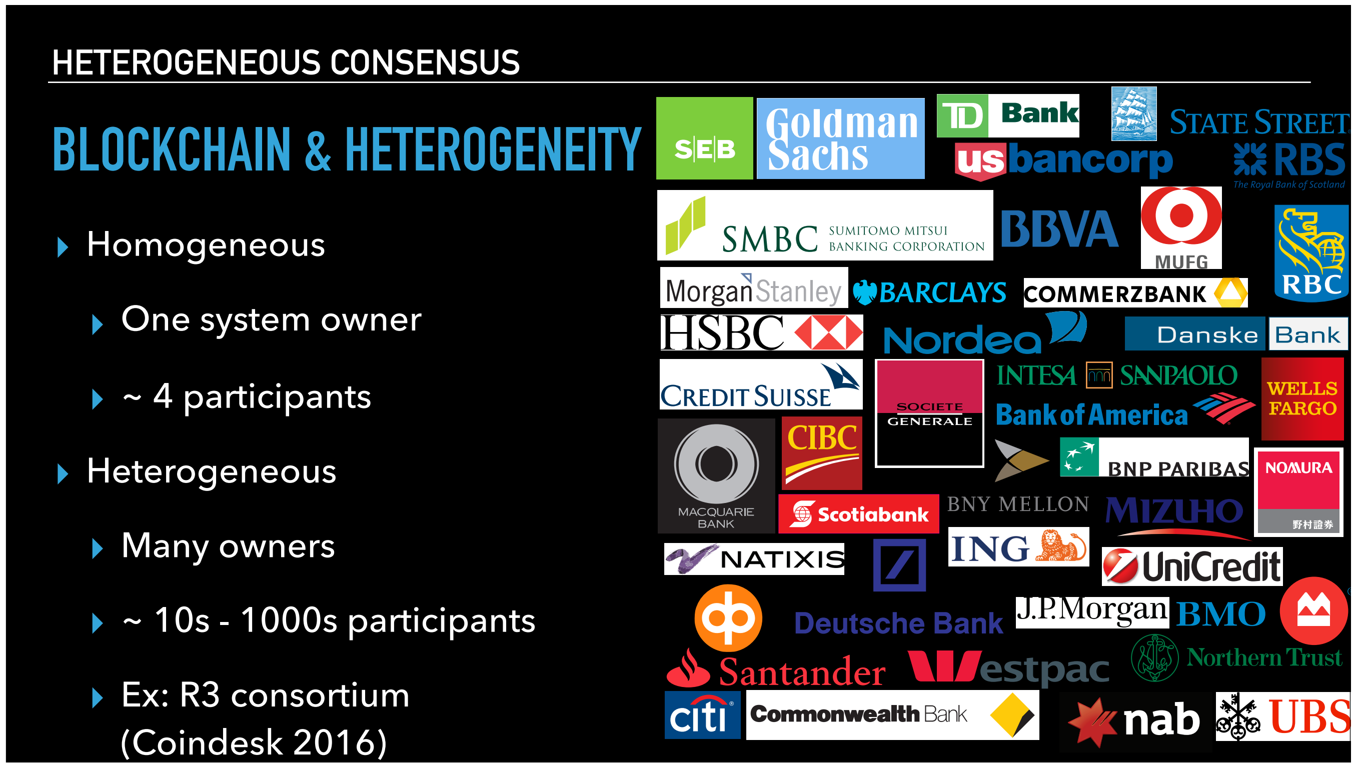
- How do we express Heterogeneous failure assumptions?
  - With an **observer graph!**
- Can we make a Consensus algorithm that works with them?
  - Iff Quorum Requirement is satisfiable
  - There's an algorithm for that



So Yes, we know exactly when we can make a consensus algorithm that works with a given graph.

We run our transitivity algorithm to condense the graph, and
we have an algorithm that produces quorums from the graph, whenever quorums exist that meet the requirement.

They may not be “the best” quorums, but if you have better ideas, we have a requirement against which you can check them.

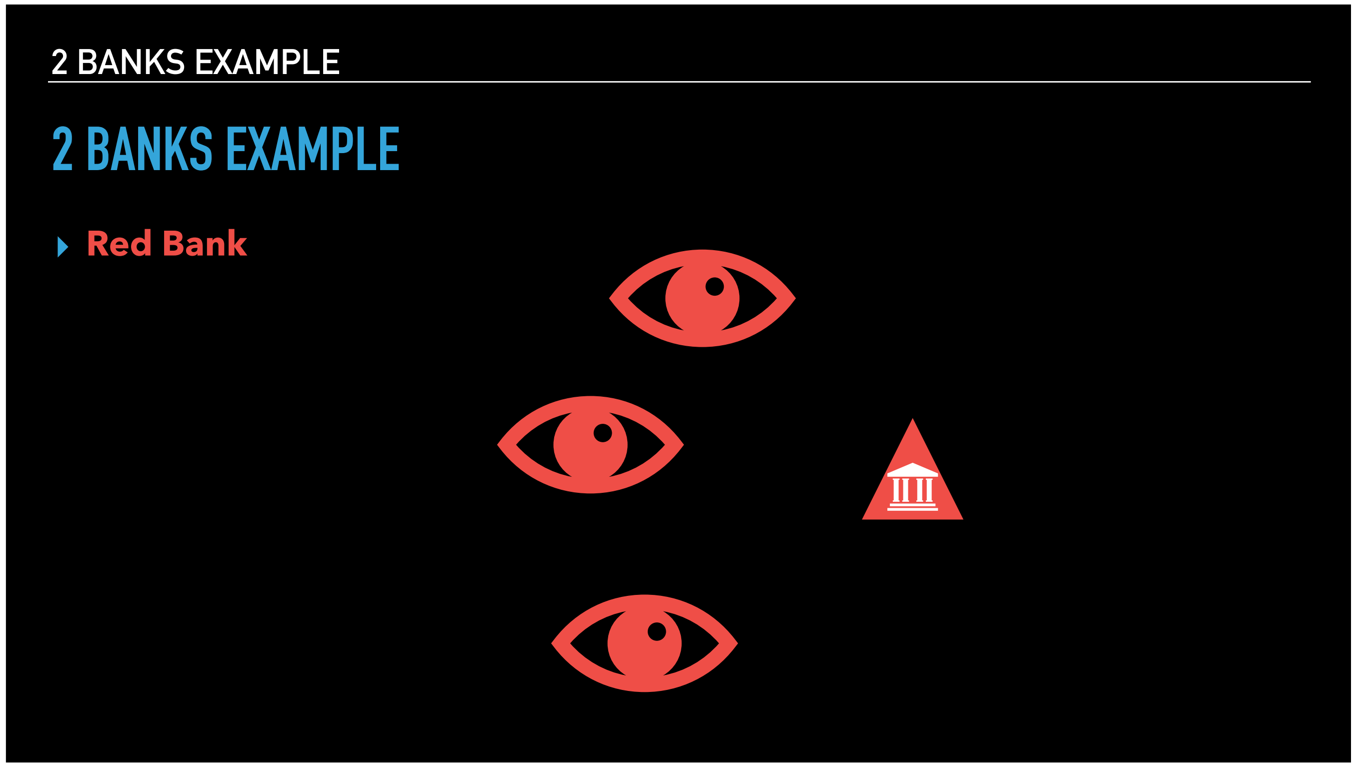


HETEROGENEOUS CONSENSUS

BLOCKCHAIN & HETEROGENEITY

- Homogeneous
  - One system owner
  - ~ 4 participants
- Heterogeneous
  - Many owners
  - ~ 10s - 1000s participants
  - Ex: R3 consortium (Coindesk 2016)

But I promised that Heterogeneous Consensus would be useful for things like bank consortiums, where all these Banks are collections of observers with different assumptions, and
Participants are running on hardware owned by the banks, and maybe some third parties.

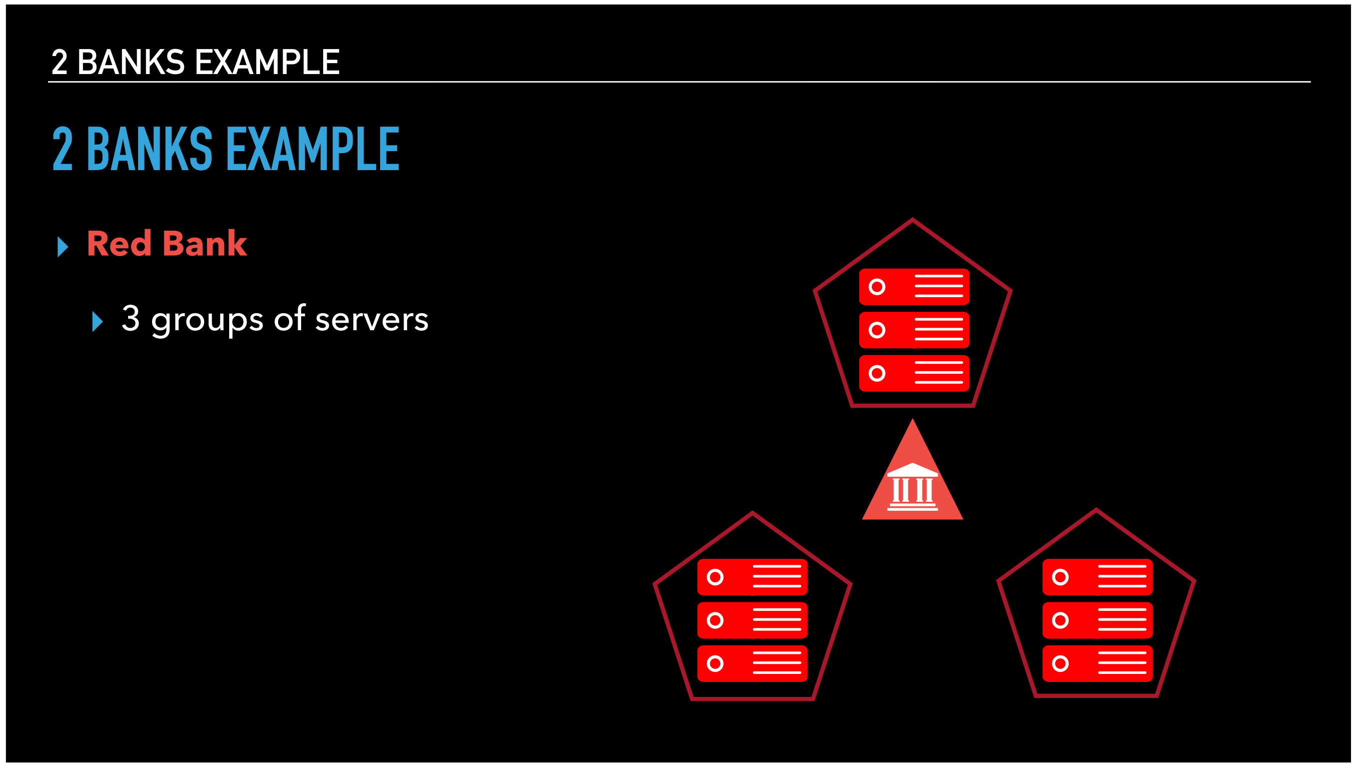


2 BANKS EXAMPLE

2 BANKS EXAMPLE

- Red Bank

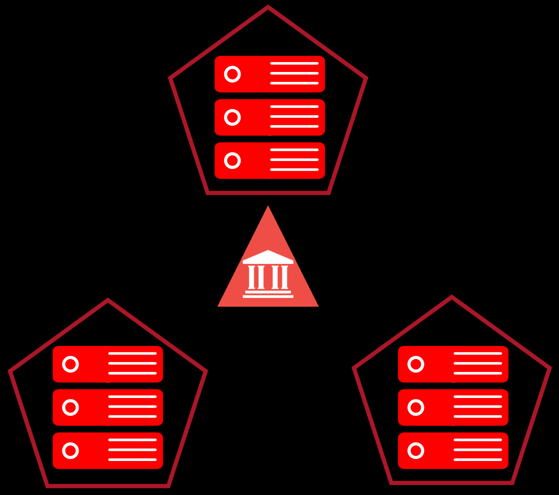
So let’s consider a small example.
We just have 2 Banks.
Red Bank is a collection of observers, namely Bankers, who want consensus on what transactions the Bank is doing.



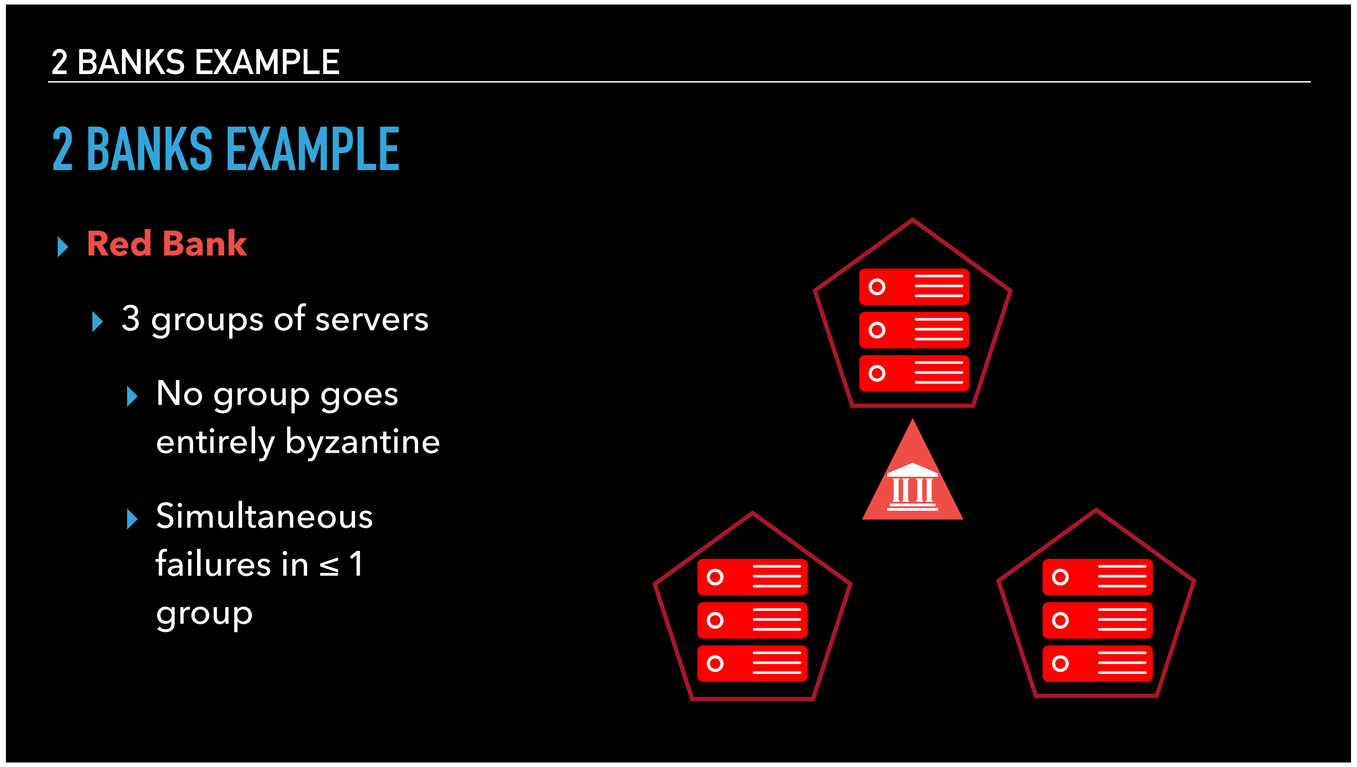
2 BANKS EXAMPLE

2 BANKS EXAMPLE

- **Red Bank**
  - 3 groups of servers



Red Bank owns three groups of servers around the world.



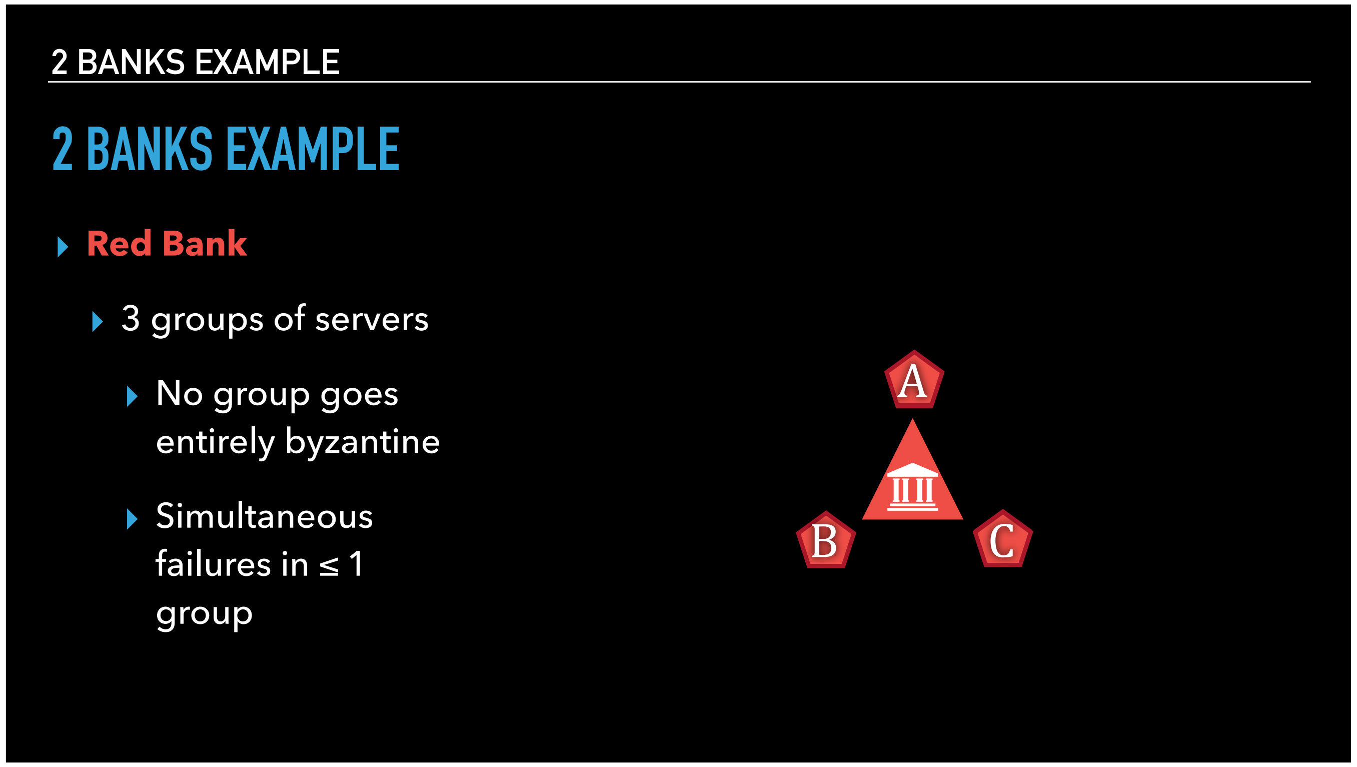
2 BANKS EXAMPLE

2 BANKS EXAMPLE

- Red Bank
  - 3 groups of servers
    - No group goes entirely byzantine
    - Simultaneous failures in ≤ 1 group

And it assumes that no one of these groups will be entirely Byzantine, and
there won't be crashes in more than one group at a time.

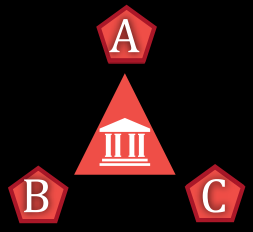
Maybe these are different branches of the bank, or datacenters on different power grids.
CLICK
For the moment, I'm just going to draw these groups as Hexegons, and elide the details of what the participants within them look like.



2 BANKS EXAMPLE

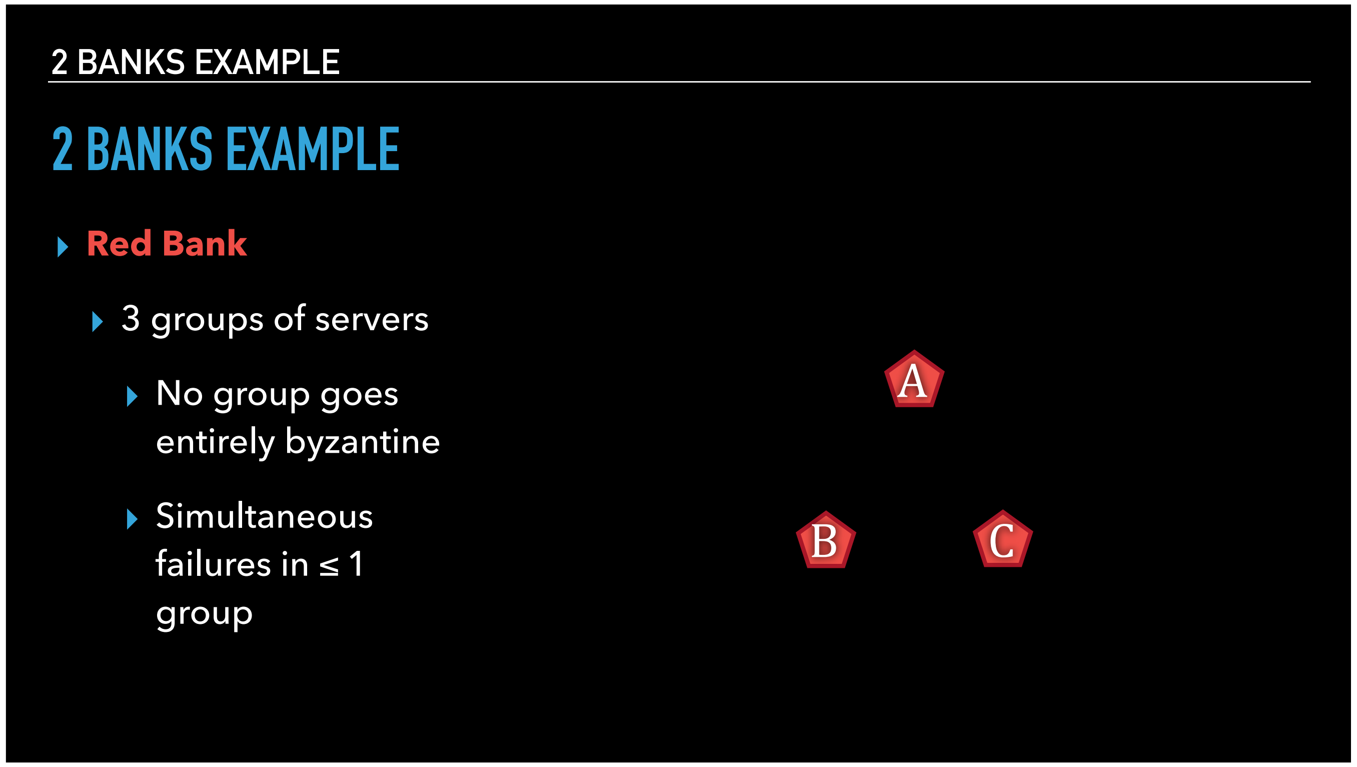
2 BANKS EXAMPLE

- Red Bank
  - 3 groups of servers
    - No group goes entirely byzantine
    - Simultaneous failures in ≤ 1 group



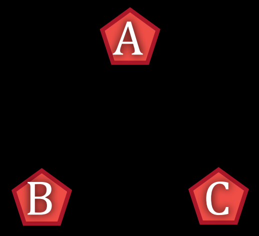
And it assumes that no one of these groups will be entirely Byzantine, and
there won't be crashes in more than one group at a time.

Maybe these are different branches of the bank, or datacenters on different power grids.
CLICK
For the moment, I'm just going to draw these groups as Hexegons, and elide the details of what the participants within them look like.

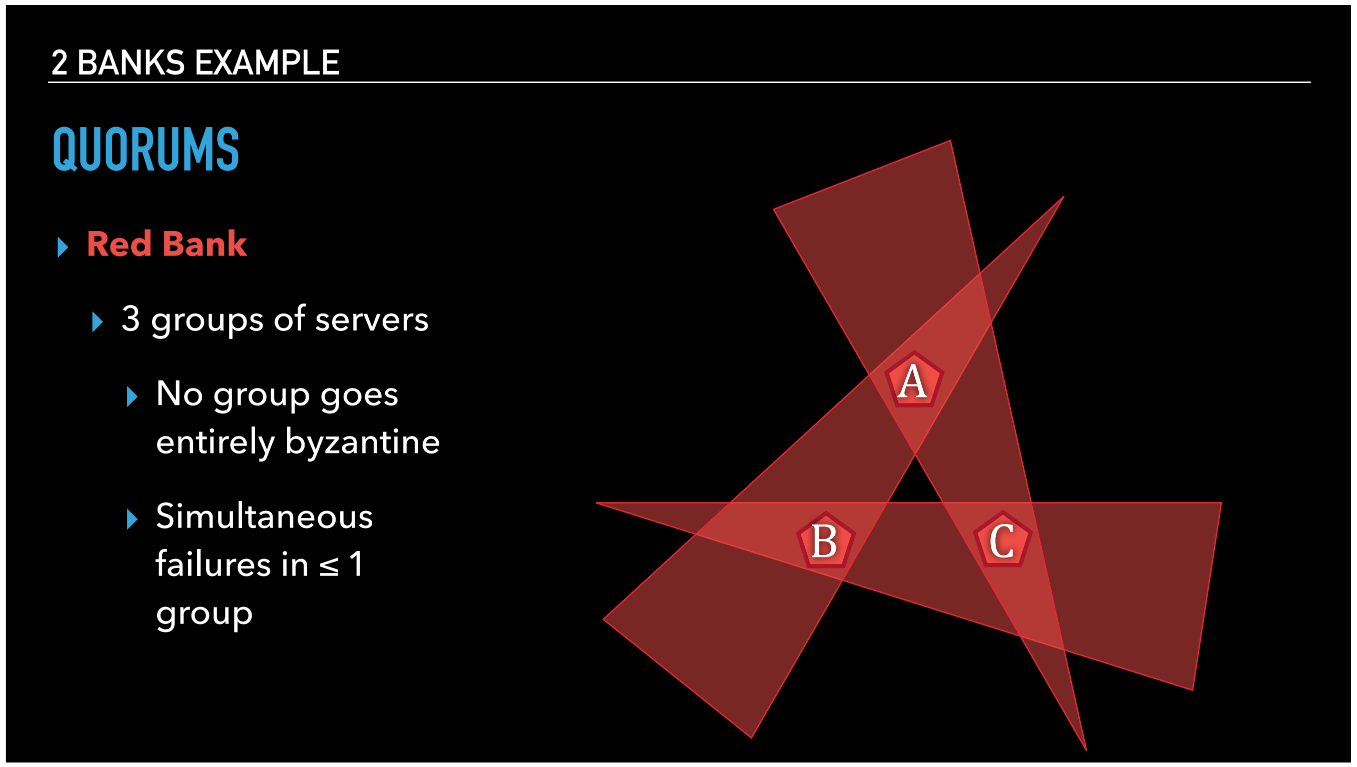


2 BANKS EXAMPLE

- **Red Bank**
  - 3 groups of servers
    - No group goes entirely byzantine
    - Simultaneous failures in ≤ 1 group



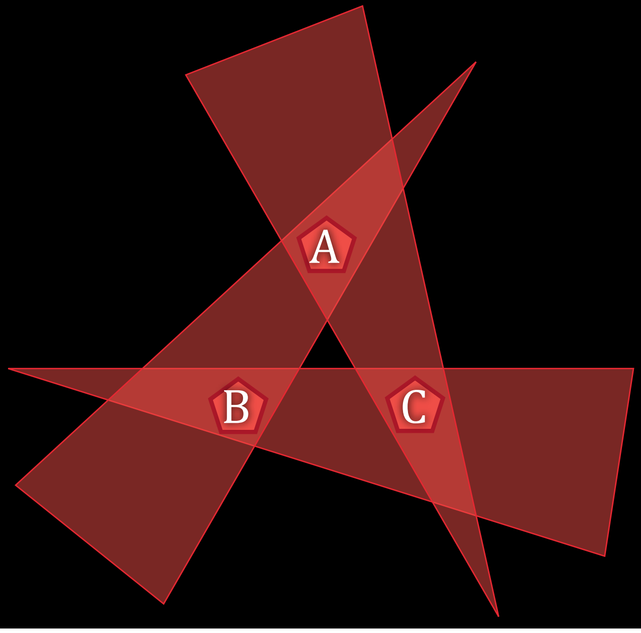
So we might want a Paxos working with these three groups of participants which satisfies the Red Bank Observers.



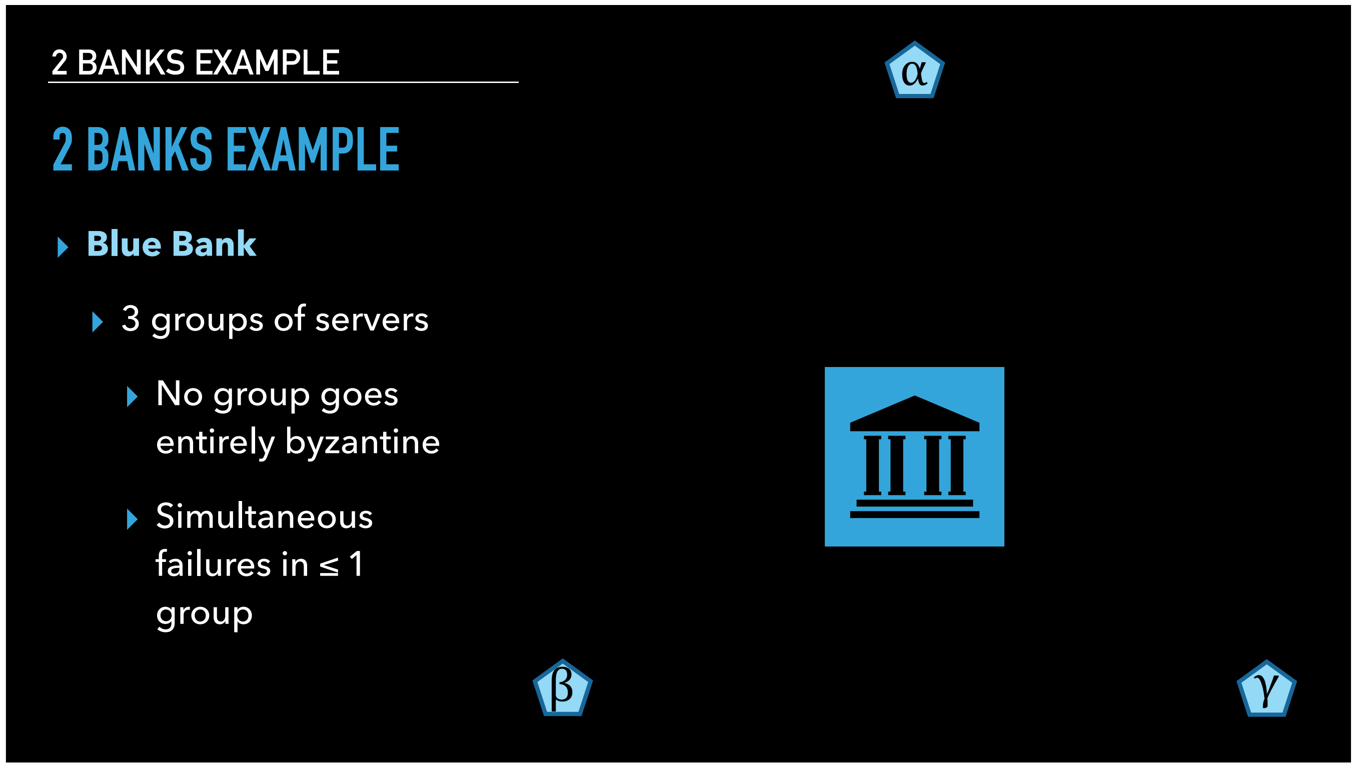
2 BANKS EXAMPLE

QUORUMS

- Red Bank
  - 3 groups of servers
    - No group goes entirely byzantine
    - Simultaneous failures in ≤ 1 group



And this is what its quorums look like.
So long as no group wholly equivocates, no two quorums can make observers decide different things, and so long as two groups are failure-free, at least one quorum can make the observers decide.

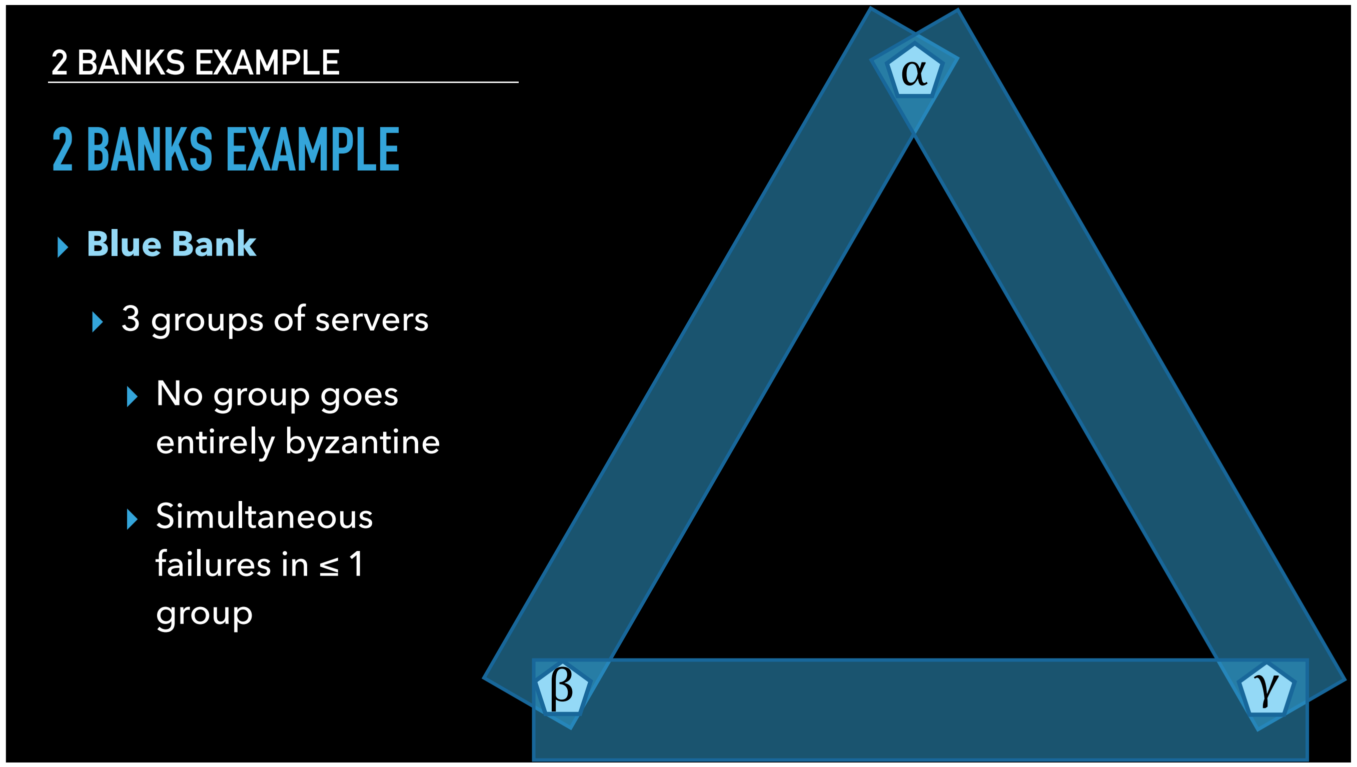


2 BANKS EXAMPLE

2 BANKS EXAMPLE

- **Blue Bank**
  - 3 groups of servers
    - No group goes entirely byzantine
    - Simultaneous failures in ≤ 1 group

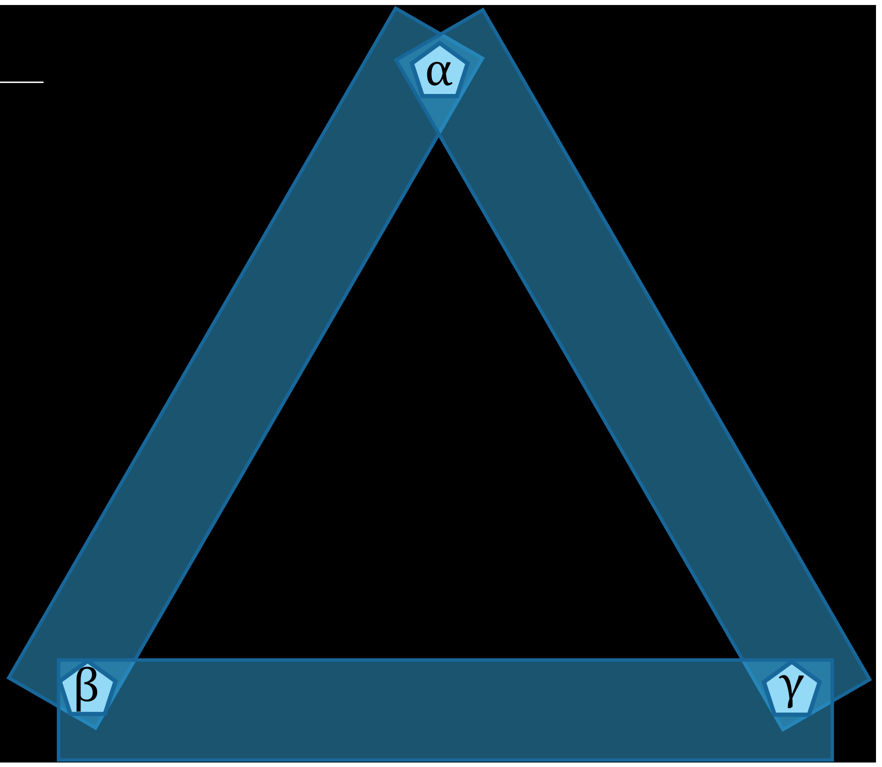
Blue Bank has a similar setup, with three groups of blue-bank participants.



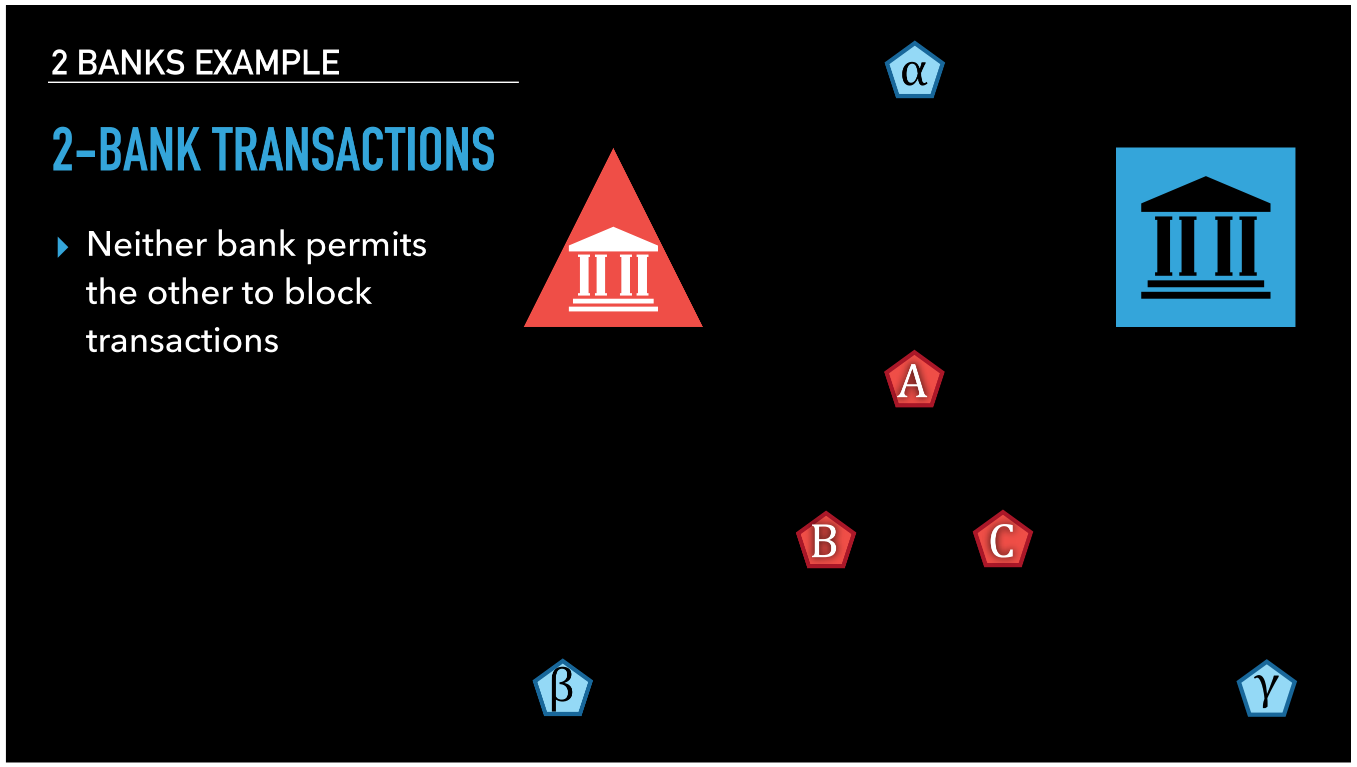
2 BANKS EXAMPLE

2 BANKS EXAMPLE

- **Blue Bank**
  - 3 groups of servers
    - No group goes entirely byzantine
    - Simultaneous failures in ≤ 1 group



And if they were to run Consensus alone, these are what their quorums would be.



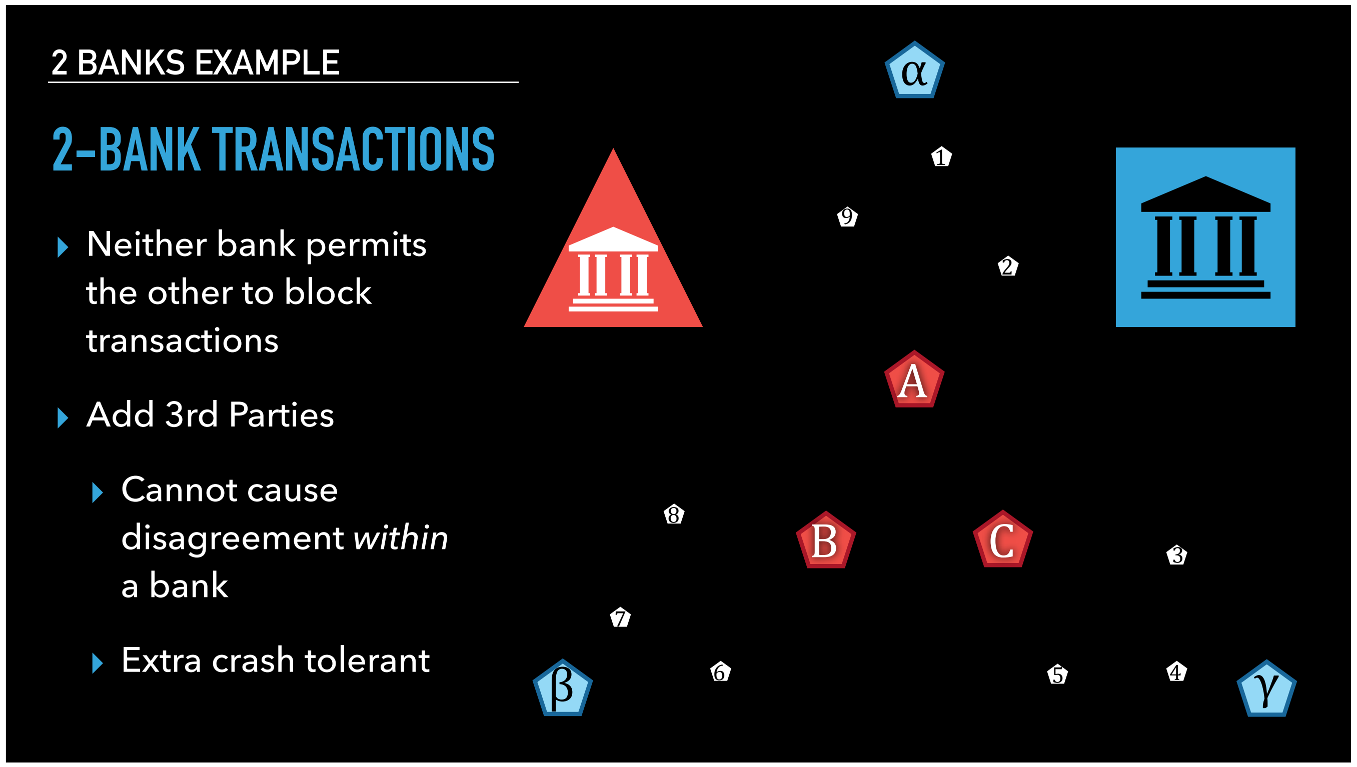
2 BANKS EXAMPLE

2-BANK TRANSACTIONS

- Neither bank permits the other to block transactions

But let's say the Two banks want to run consensus together.
Maybe they want to be able to agree on inter-bank transactions.

However, they don't trust each other.
Neither bank wants the other's participants to be able to hold up its own transactions.
And they especially don't want any outsiders to force disagreements within a bank.



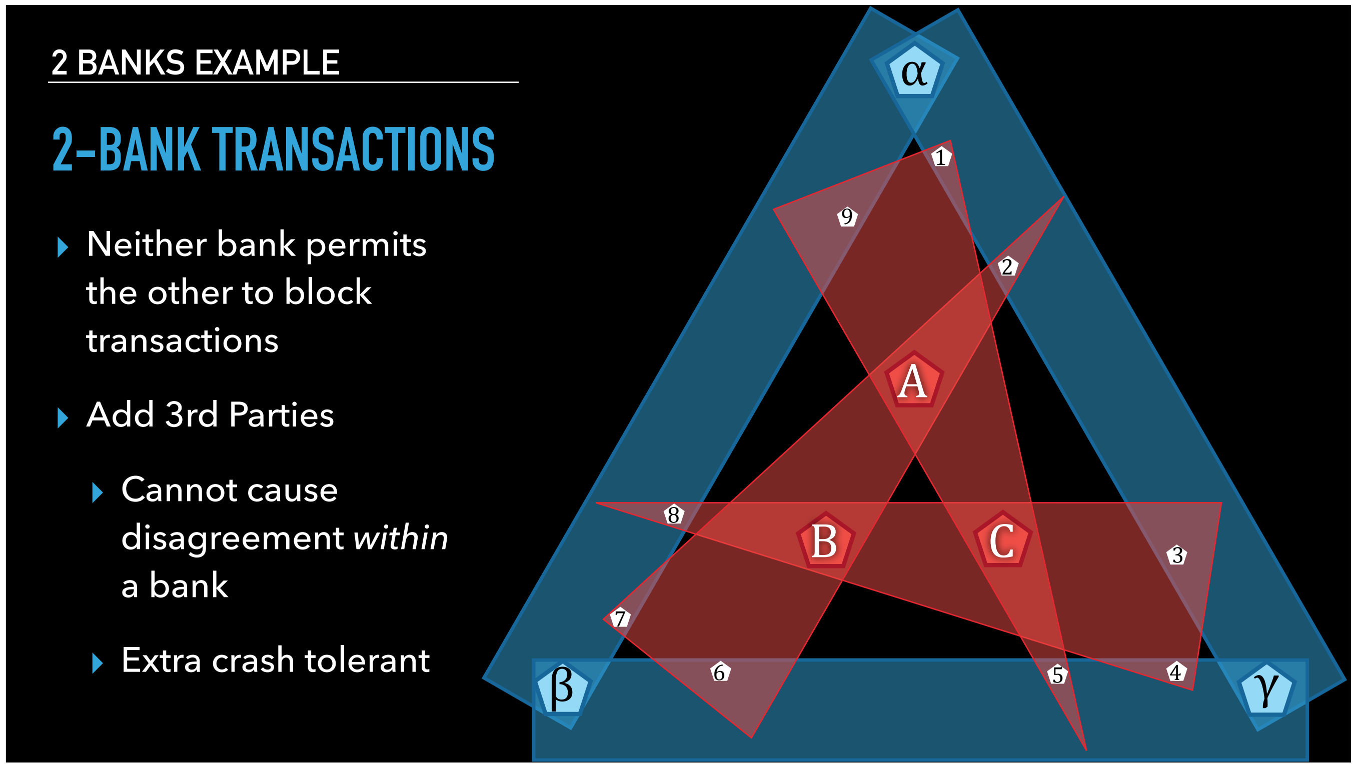
2 BANKS EXAMPLE

2-BANK TRANSACTIONS

- Neither bank permits the other to block transactions
- Add 3rd Parties
  - Cannot cause disagreement *within* a bank
  - Extra crash tolerant

So they will have to add third parties.
They want to be more tolerant of these third parties crashing than they are of their internal servers, and they still demand that no third party can cause disagreement within a bank.

Now as it turns out, we can make a Heterogeneous Consensus for these Banks.

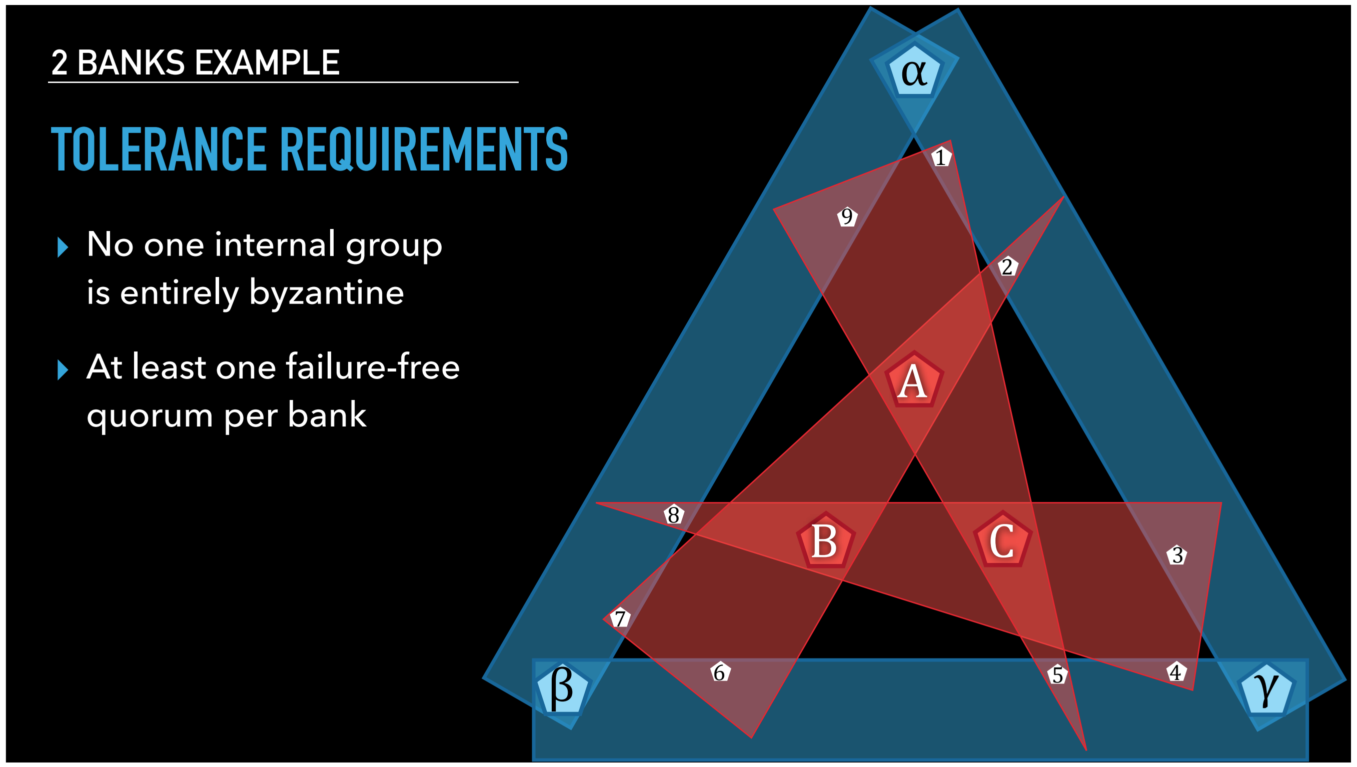


2 BANKS EXAMPLE

2-BANK TRANSACTIONS

- Neither bank permits the other to block transactions
- Add 3rd Parties
  - Cannot cause disagreement *within* a bank
  - Extra crash tolerant

These are the quorums.

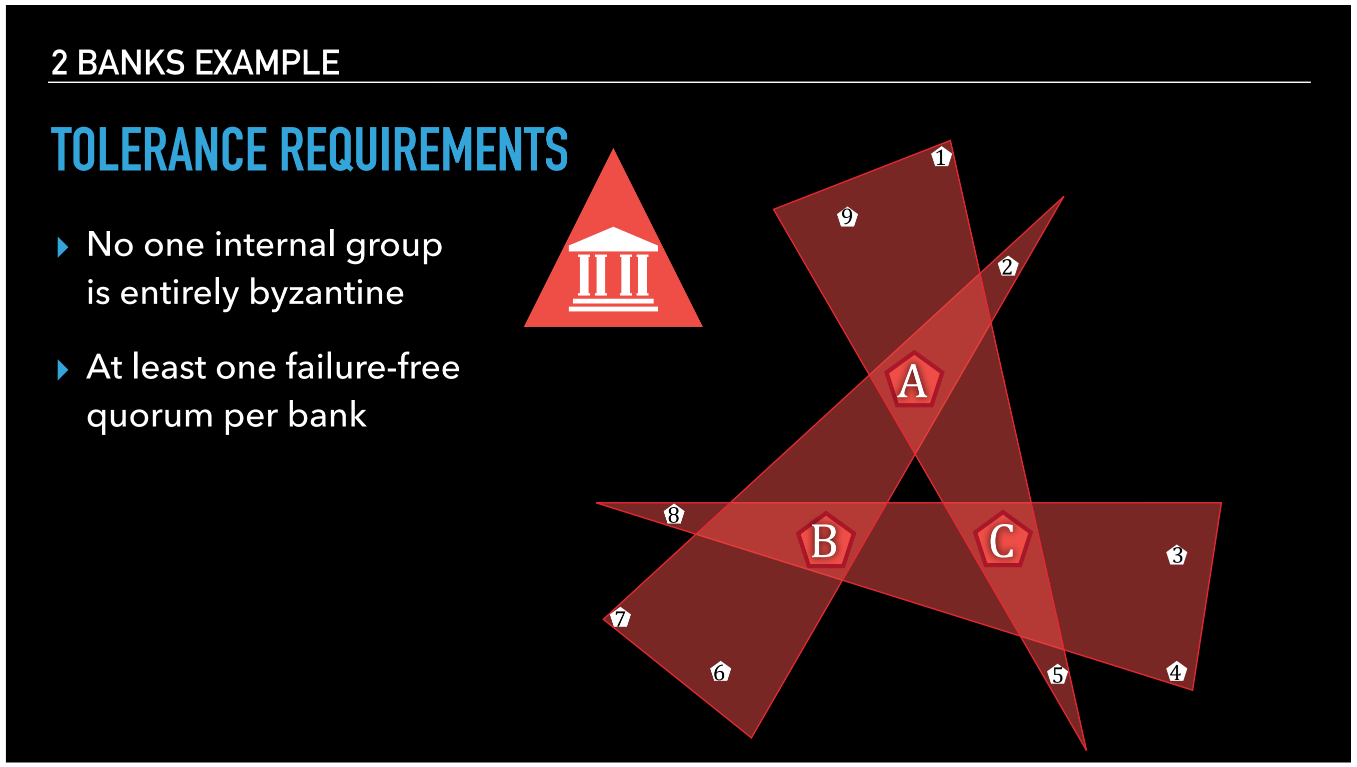


2 BANKS EXAMPLE

TOLERANCE REQUIREMENTS

- No one internal group is entirely byzantine
- At least one failure-free quorum per bank

Now, we still can't tolerate one of the bank's participant groups going entirely byzantine, and we still require a failure-free quorum per bank.



2 BANKS EXAMPLE

TOLERANCE REQUIREMENTS

- No one internal group is entirely byzantine
- At least one failure-free quorum per bank

A
B
C
1
2
3
4
5
6
7
8
9

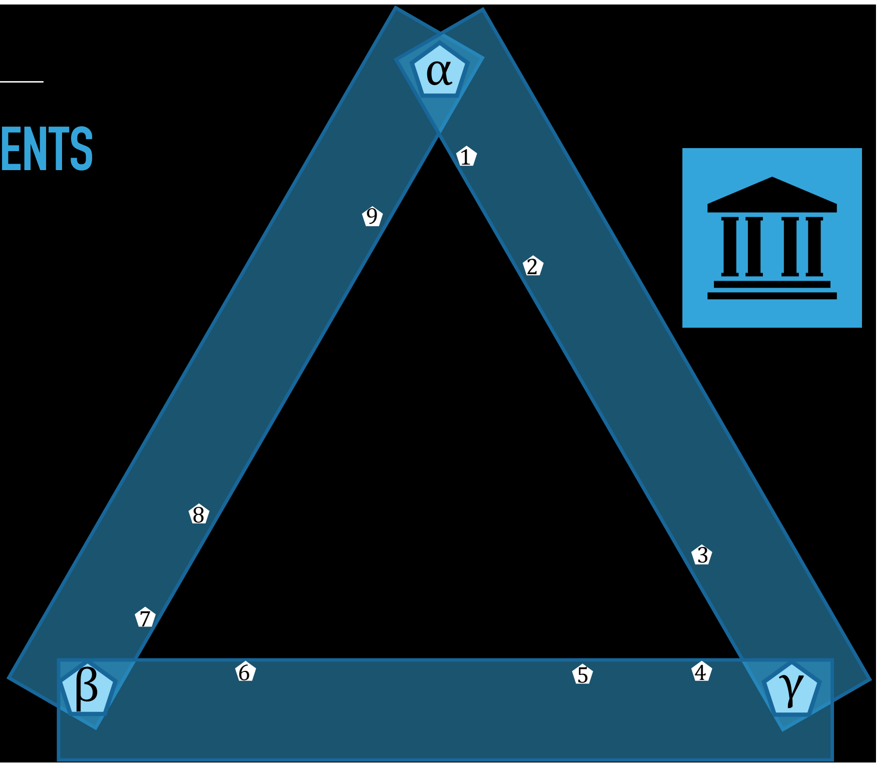
In Particular, for Red Bank’s observers, the quorums look like this.

We still require that no one internal group is entirely Byzantine, and we are going to need at least one quorum to be failure-free to decide.

2 BANKS EXAMPLE

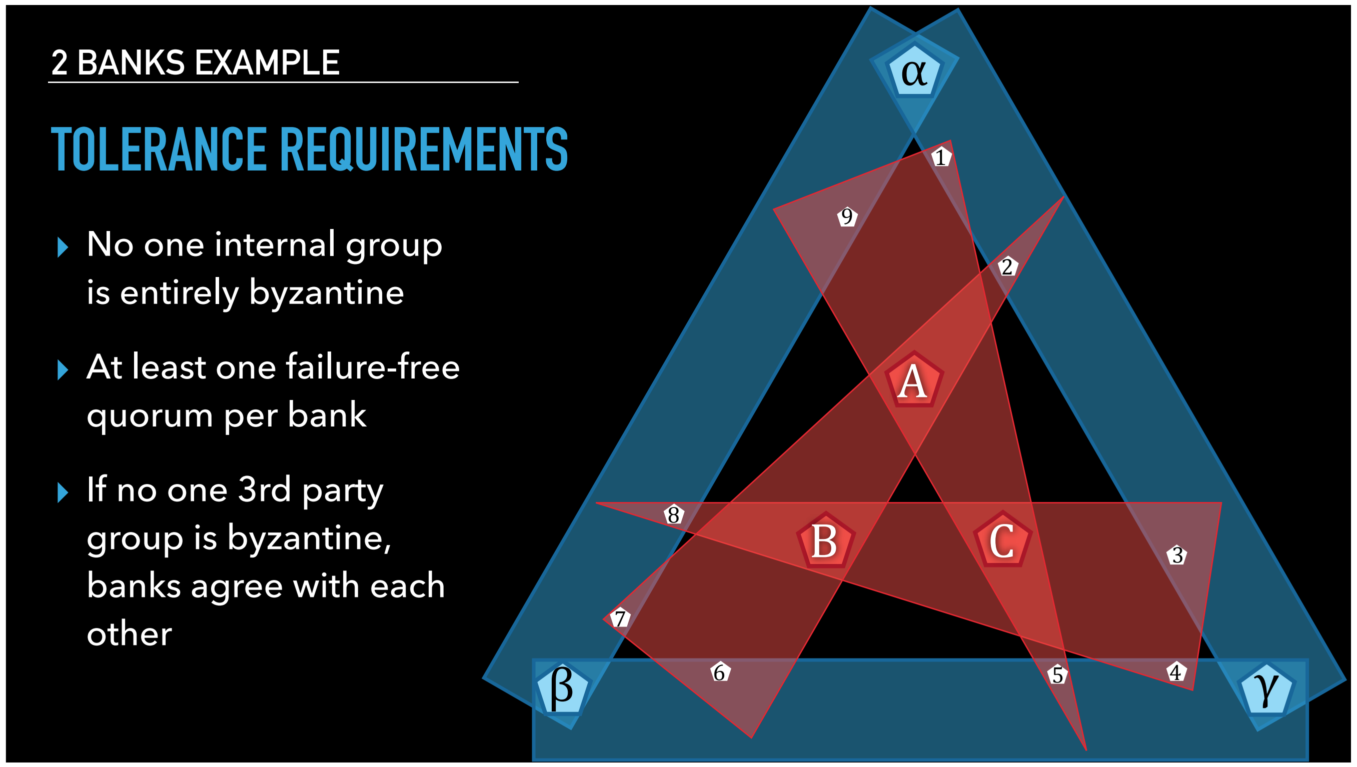
TOLERANCE REQUIREMENTS

- No one internal group is entirely byzantine
- At least one failure-free quorum per bank



Blue Bank has a similar setup.

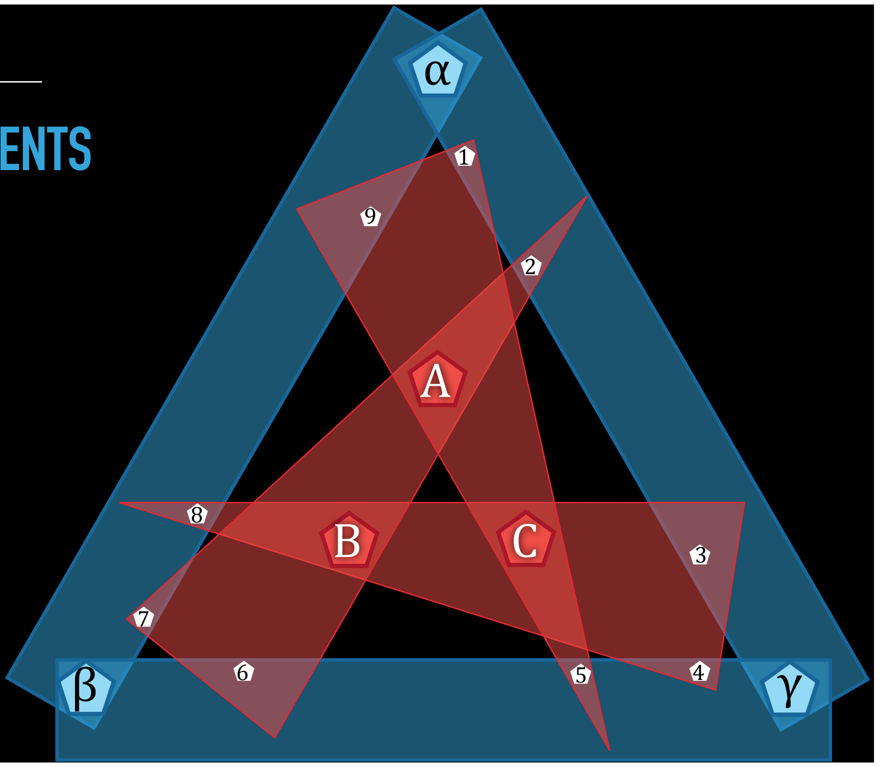
We still require that no one internal group is entirely Byzantine, and we are going to need at least one quorum to be failure-free to decide.



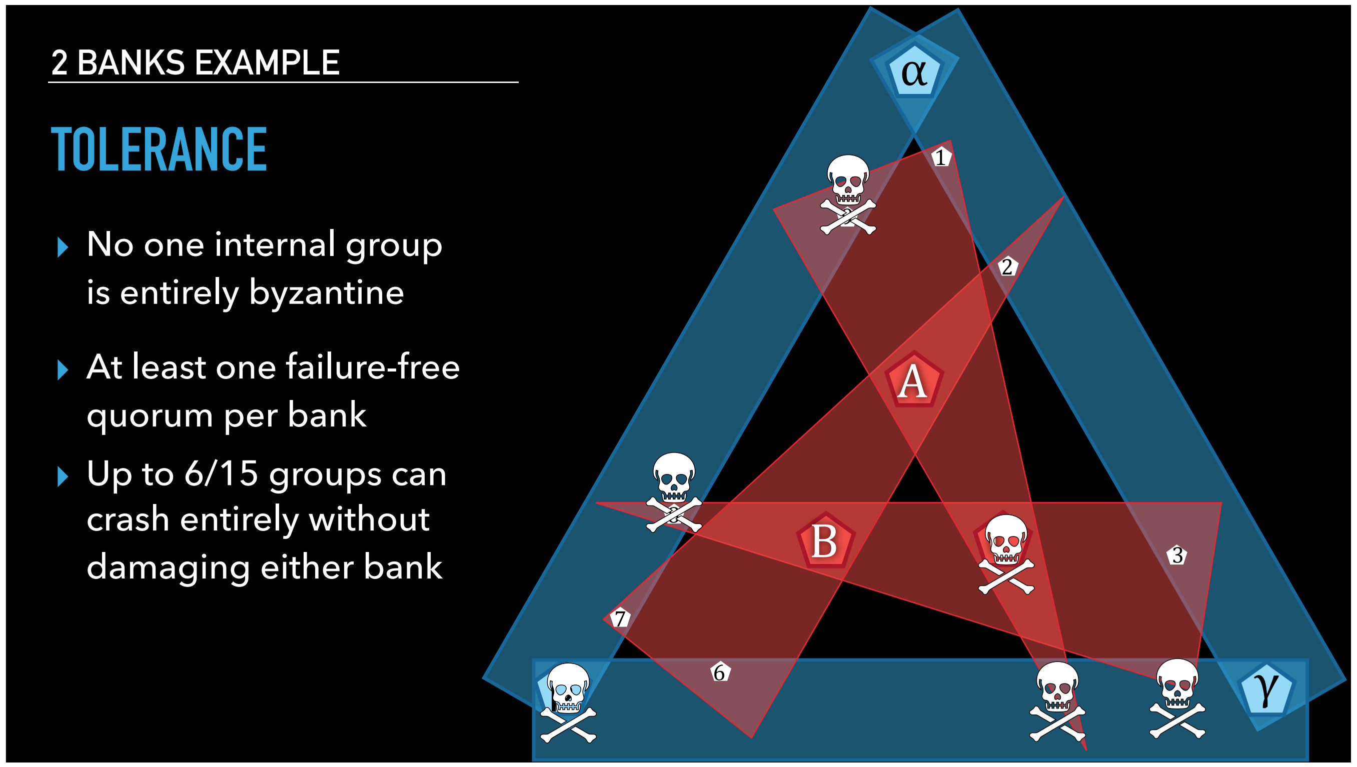
2 BANKS EXAMPLE

TOLERANCE REQUIREMENTS

- No one internal group is entirely byzantine
- At least one failure-free quorum per bank
- If no one 3rd party group is byzantine, banks agree with each other



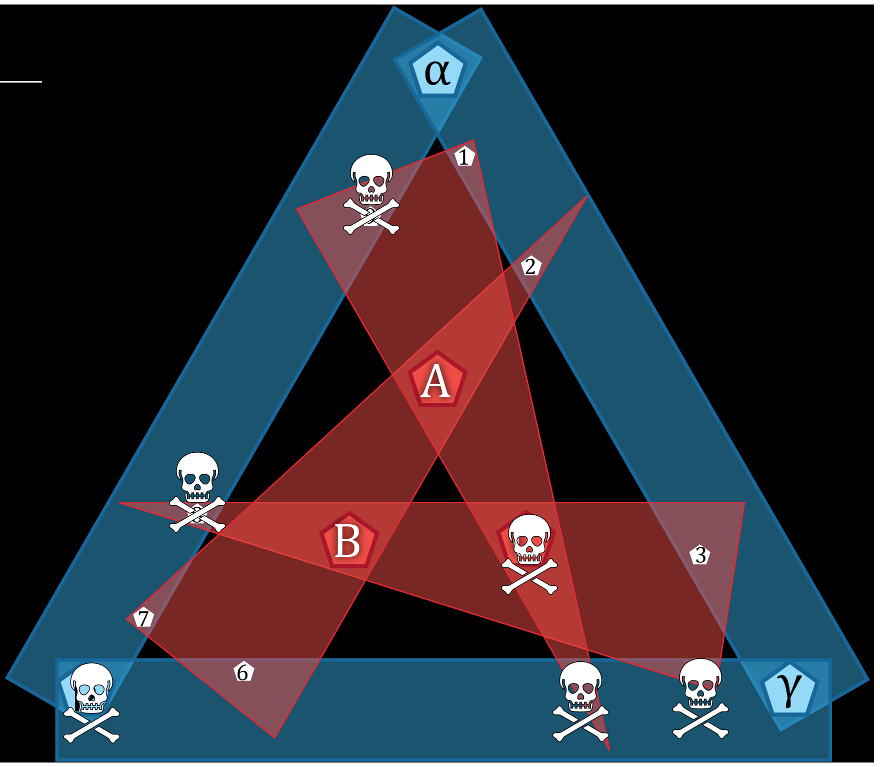
So in total, if no one bank group is entirely Byzantine, and one quorum survives per bank, the Banks always decide, and agree internally.
In addition,
If no one 3rd party group is entirely byzantine, the banks are guaranteed to agree with each other.



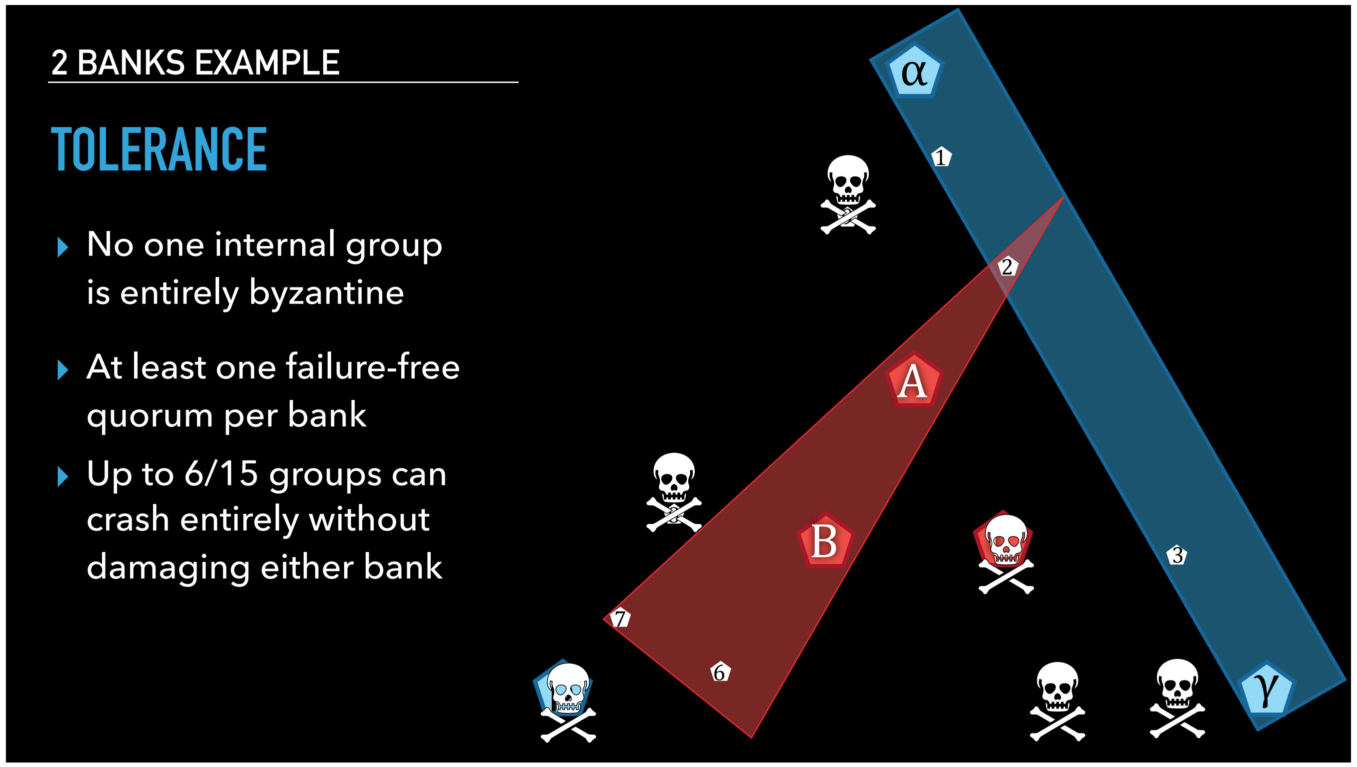
2 BANKS EXAMPLE

TOLERANCE

- No one internal group is entirely byzantine
- At least one failure-free quorum per bank
- Up to 6/15 groups can crash entirely without damaging either bank



Which means that up to 6 out of 15 of the groups could crash entirely without damaging either bank.



2 BANKS EXAMPLE

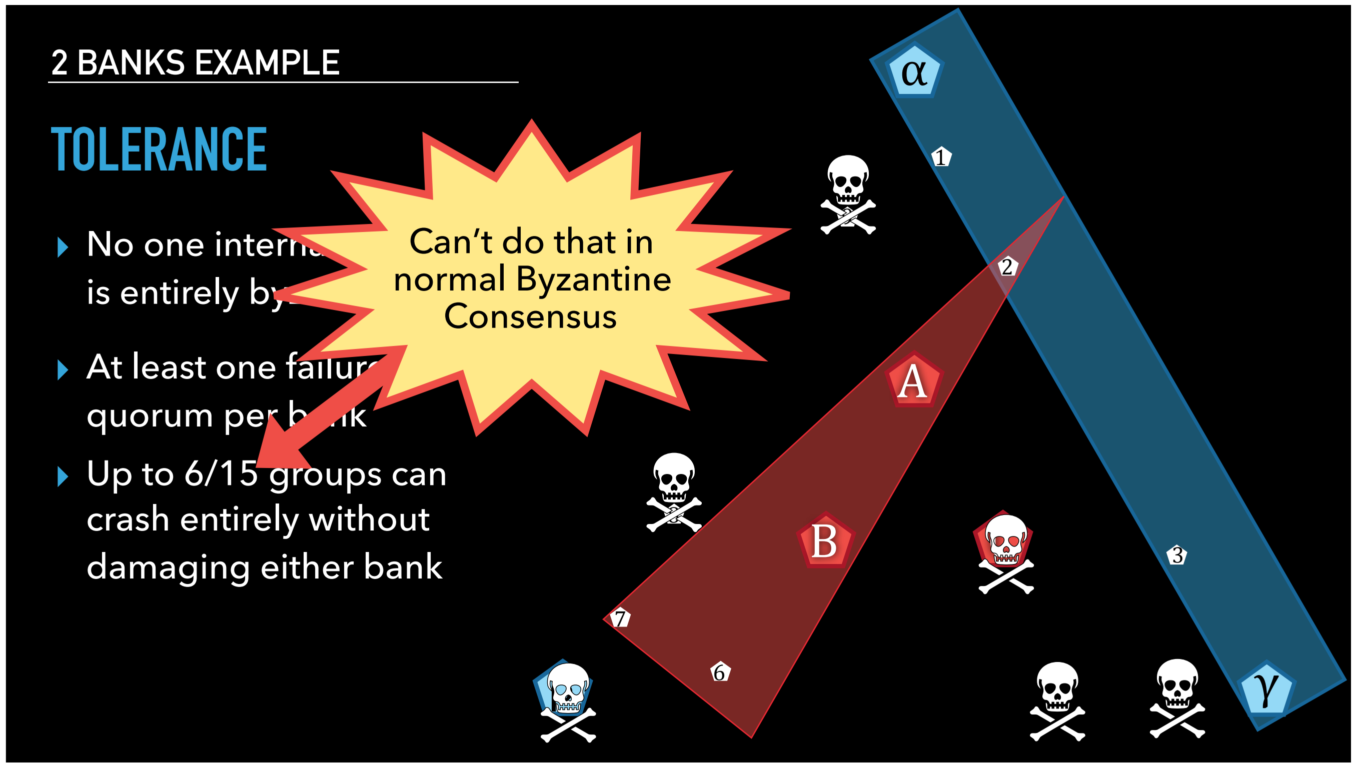
TOLERANCE

- No one internal group is entirely byzantine
- At least one failure-free quorum per bank
- Up to 6/15 groups can crash entirely without damaging either bank

There will still be a safe, live quorum for each of them, with a safe intersection.

CLICK

Now we’ve already done something that the Byzantine Consensus algorithms usually proposed for Blockchains can’t do.
They could never tolerate six out of 15 failures. The maximum is strictly less than one third.



2 BANKS EXAMPLE

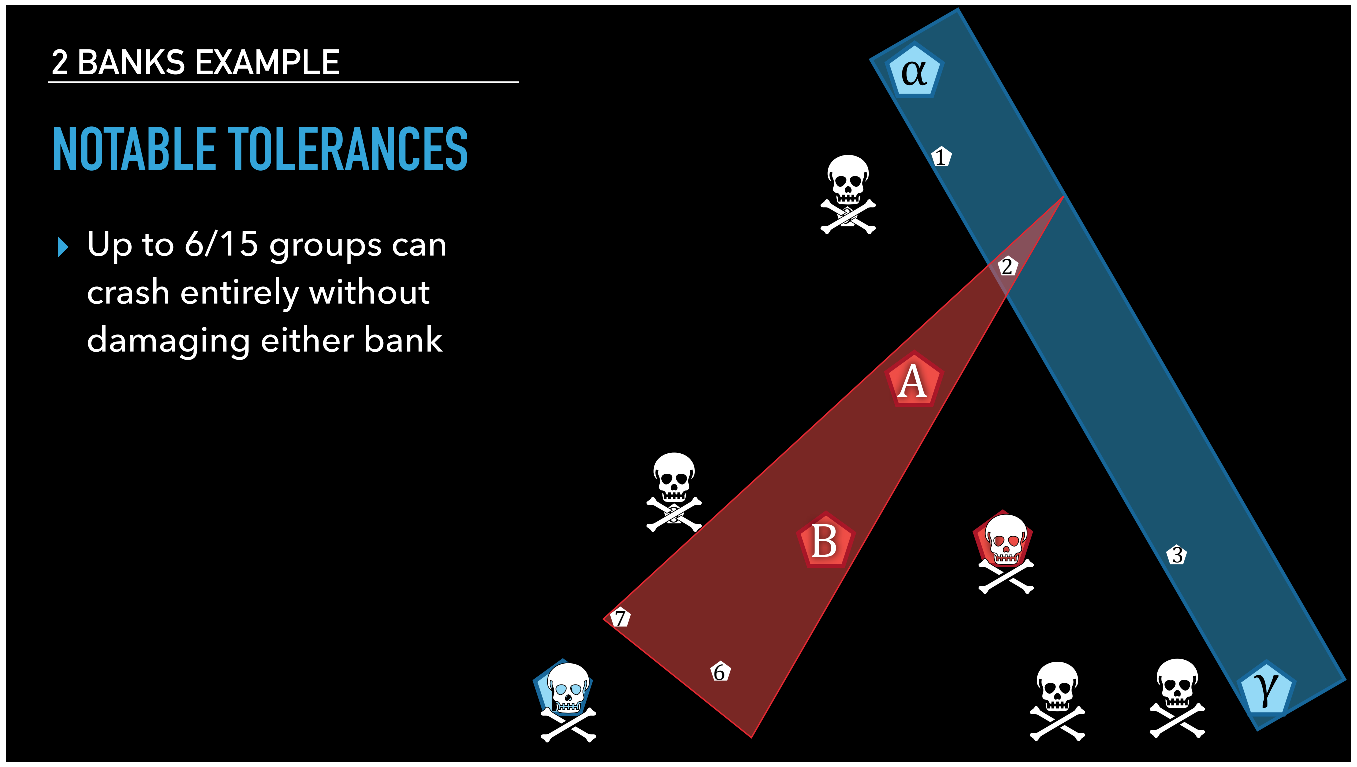
TOLERANCE

- No one interna… is entirely by…
- At least one failur… quorum per b…nk
- Up to 6/15 groups can crash entirely without damaging either bank

There will still be a safe, live quorum for each of them, with a safe intersection.

CLICK

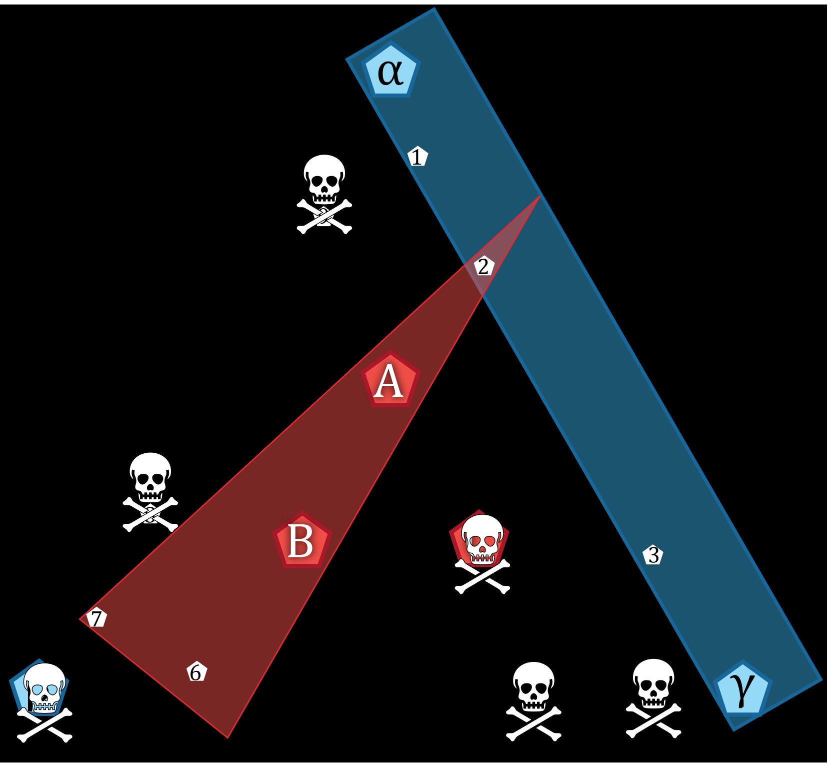
Now we've already done something that the Byzantine Consensus algorithms usually proposed for Blockchains can't do.
They could never tolerate six out of 15 failures. The maximum is strictly less than one third.



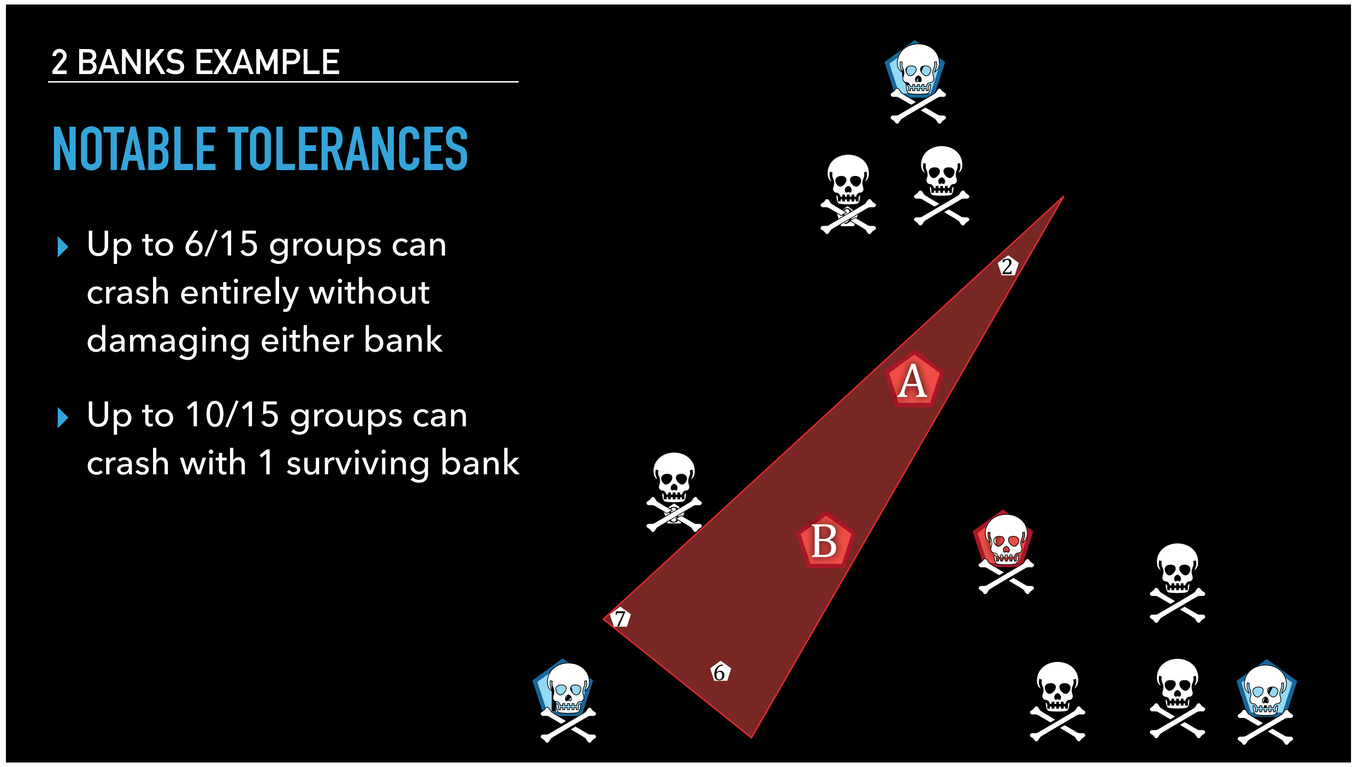
2 BANKS EXAMPLE

NOTABLE TOLERANCES

- Up to 6/15 groups can crash entirely without damaging either bank



But there are other notable failure tolerances for this Consensus.



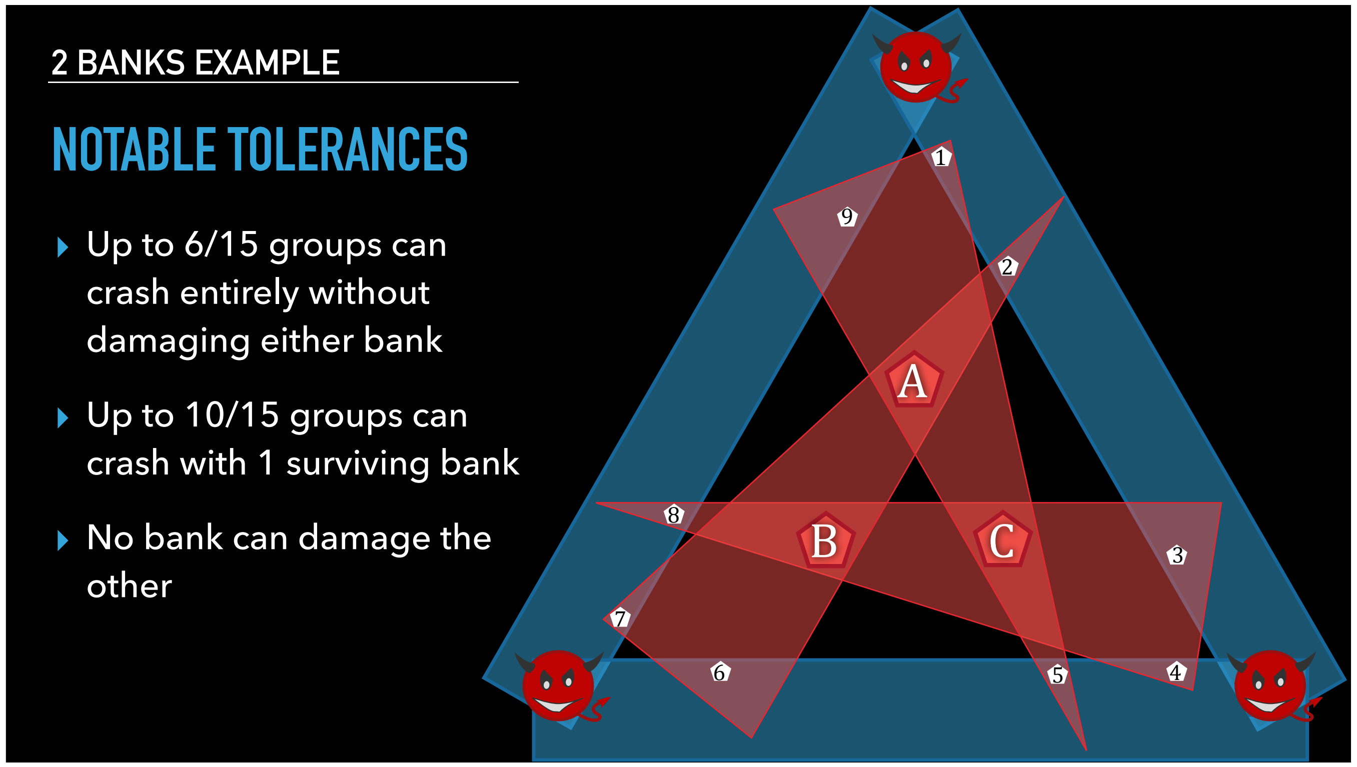
2 BANKS EXAMPLE

NOTABLE TOLERANCES

- Up to 6/15 groups can crash entirely without damaging either bank
- Up to 10/15 groups can crash with 1 surviving bank

As many as 10 out of 15 groups of participants can crash, and still leave one bank undamaged.

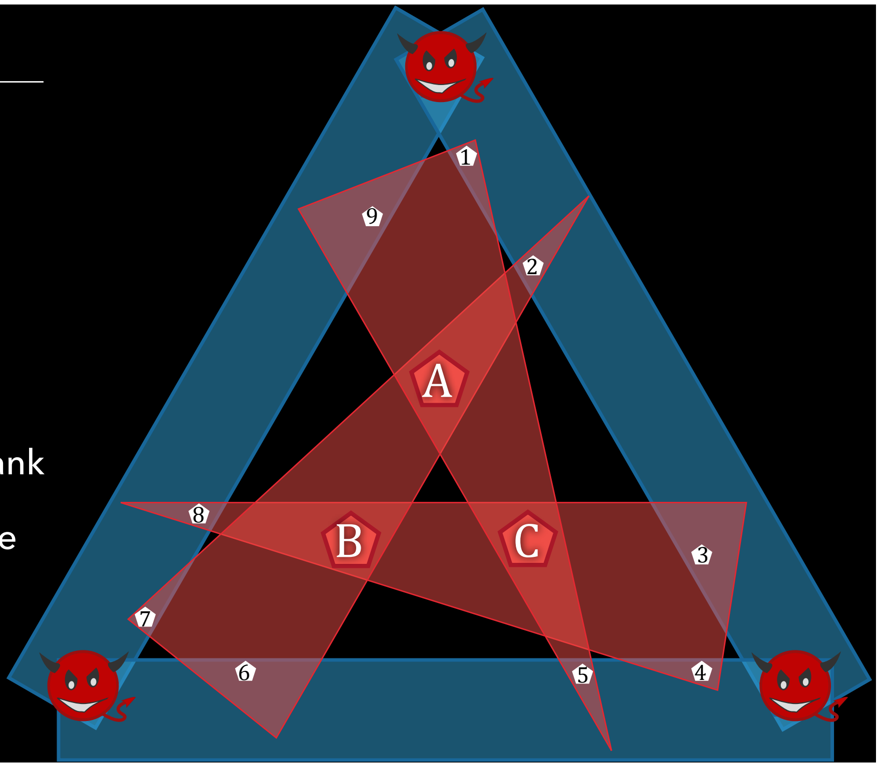
It could still have a surviving quorum.



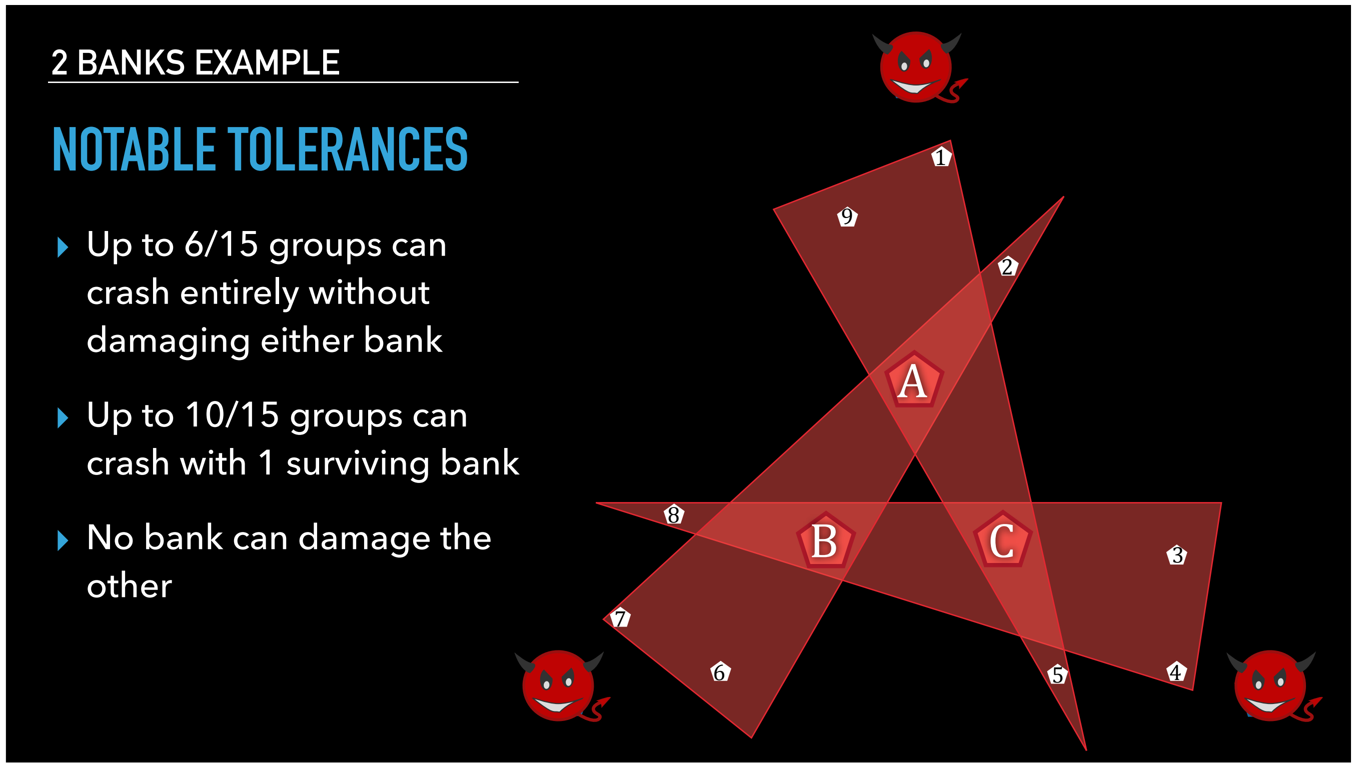
2 BANKS EXAMPLE

NOTABLE TOLERANCES

- Up to 6/15 groups can crash entirely without damaging either bank
- Up to 10/15 groups can crash with 1 surviving bank
- No bank can damage the other



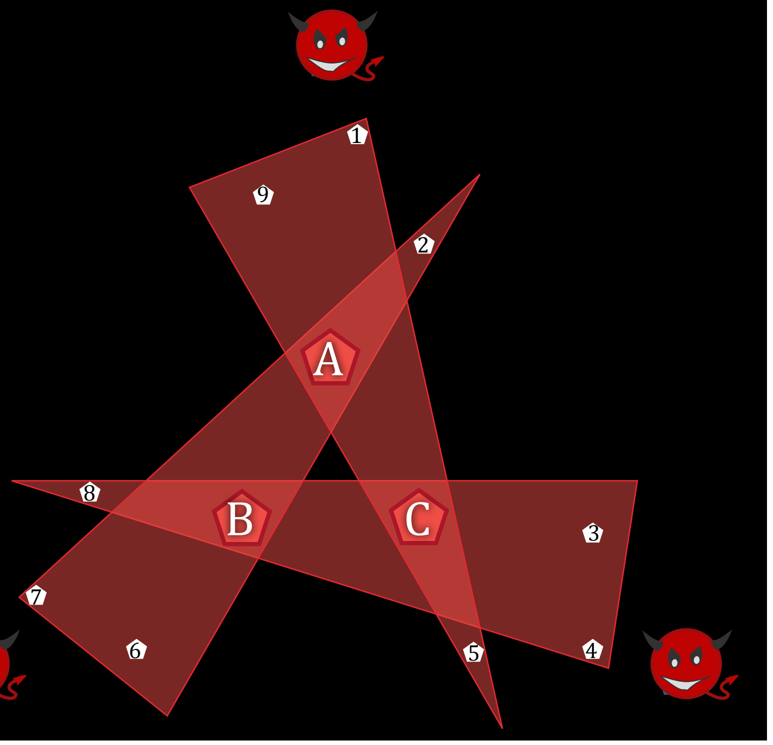
As demanded, neither bank can damage the other.



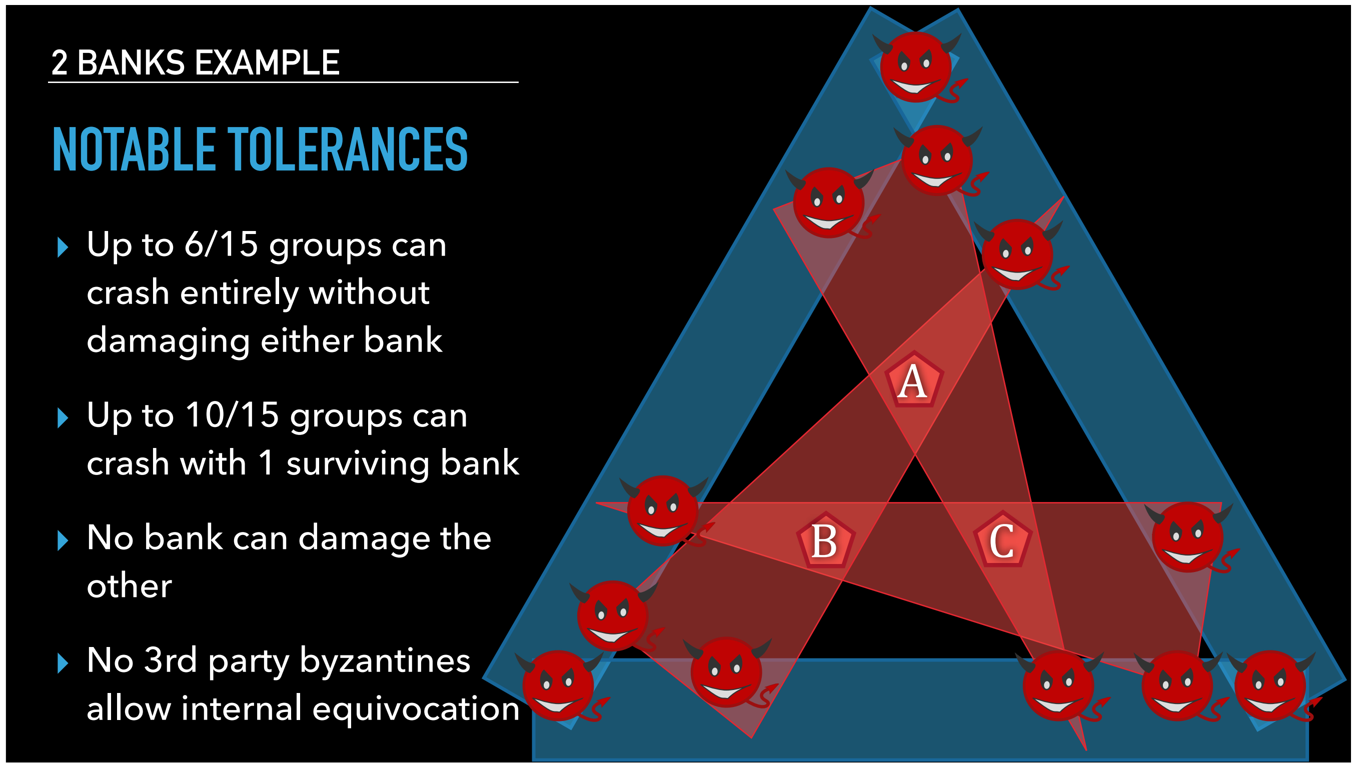
2 BANKS EXAMPLE

NOTABLE TOLERANCES

- Up to 6/15 groups can crash entirely without damaging either bank
- Up to 10/15 groups can crash with 1 surviving bank
- No bank can damage the other



In fact, each bank's participants aren't even featured in the other Bank's Paxos.

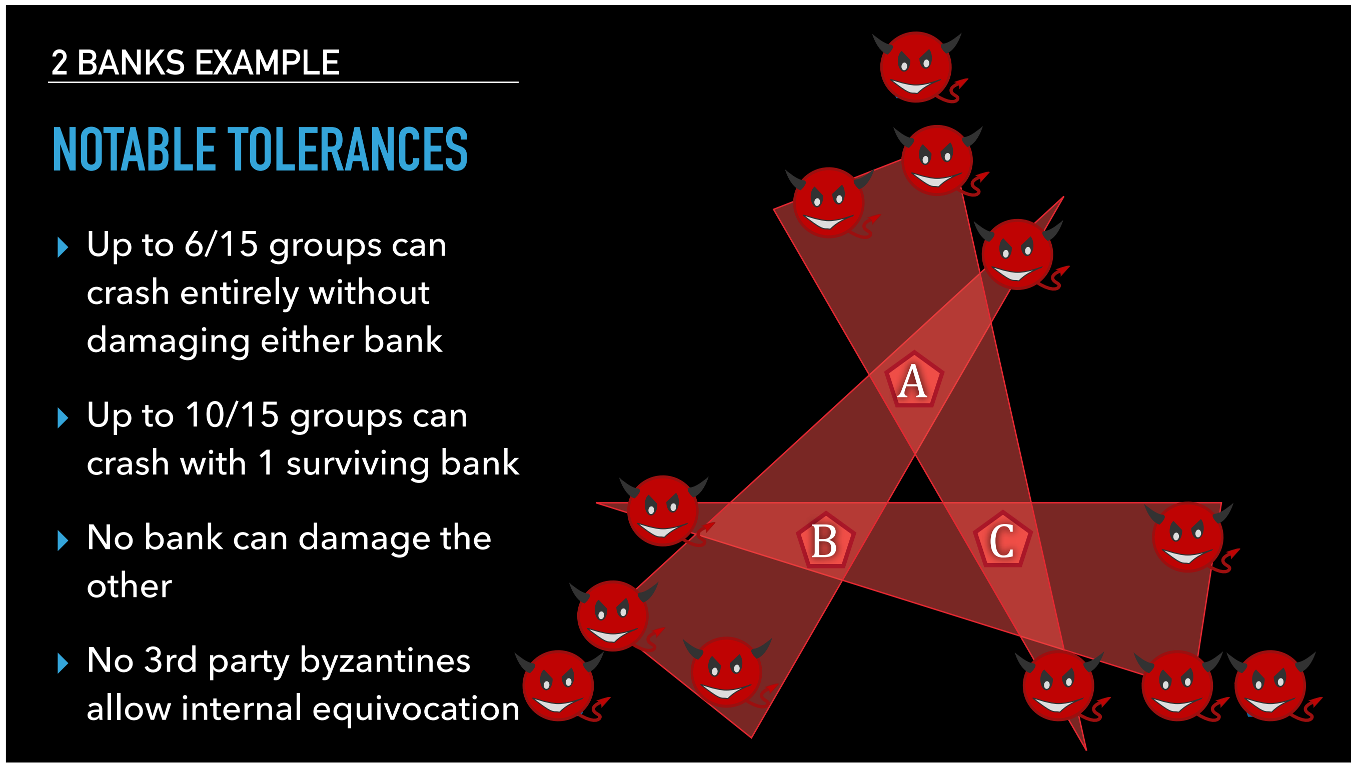


2 BANKS EXAMPLE

NOTABLE TOLERANCES

- Up to 6/15 groups can crash entirely without damaging either bank
- Up to 10/15 groups can crash with 1 surviving bank
- No bank can damage the other
- No 3rd party byzantines allow internal equivocation

And even if all the third parties and one bank are all Byzantine,

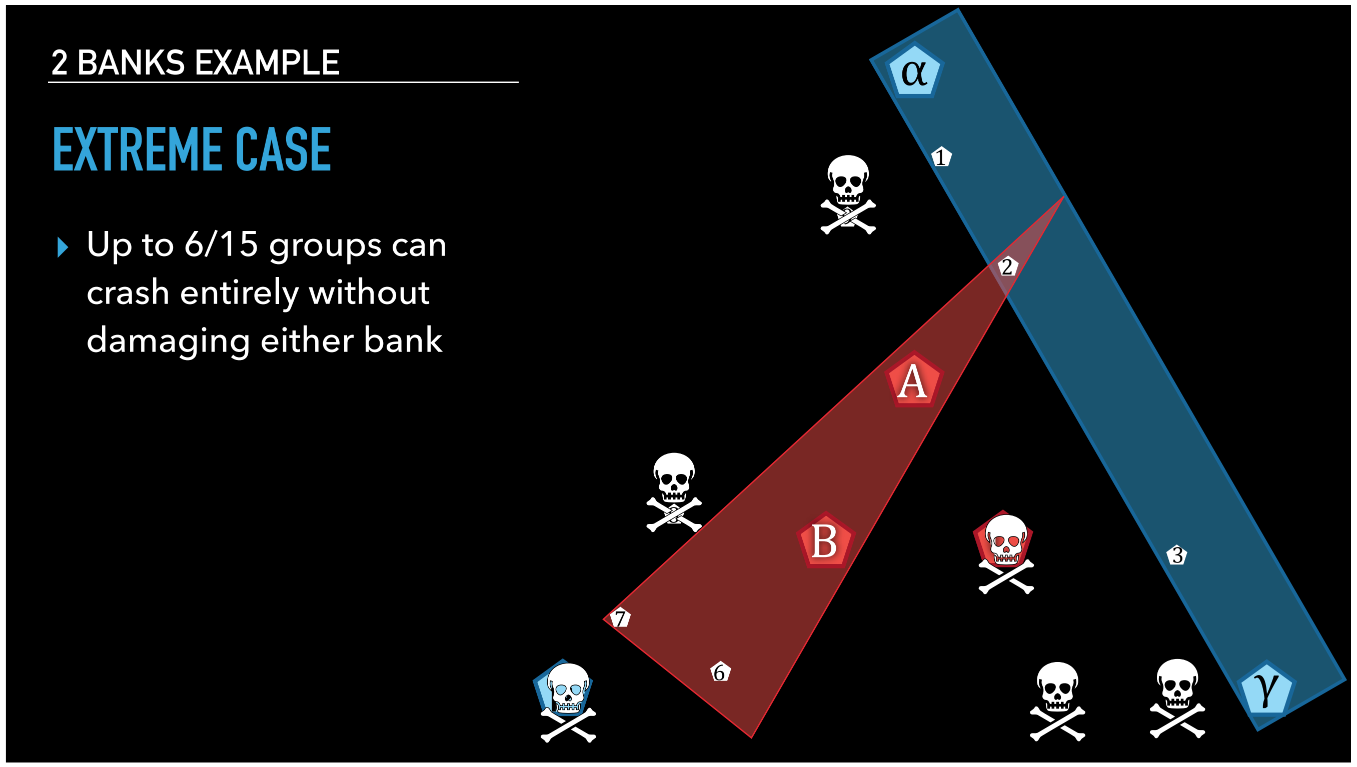


2 BANKS EXAMPLE

NOTABLE TOLERANCES

- Up to 6/15 groups can crash entirely without damaging either bank
- Up to 10/15 groups can crash with 1 surviving bank
- No bank can damage the other
- No 3rd party byzantines allow internal equivocation

the surviving bank’s quorums will have a safe intersection,
So banks can never have internal equivocation unless their own participants are Byzantine.



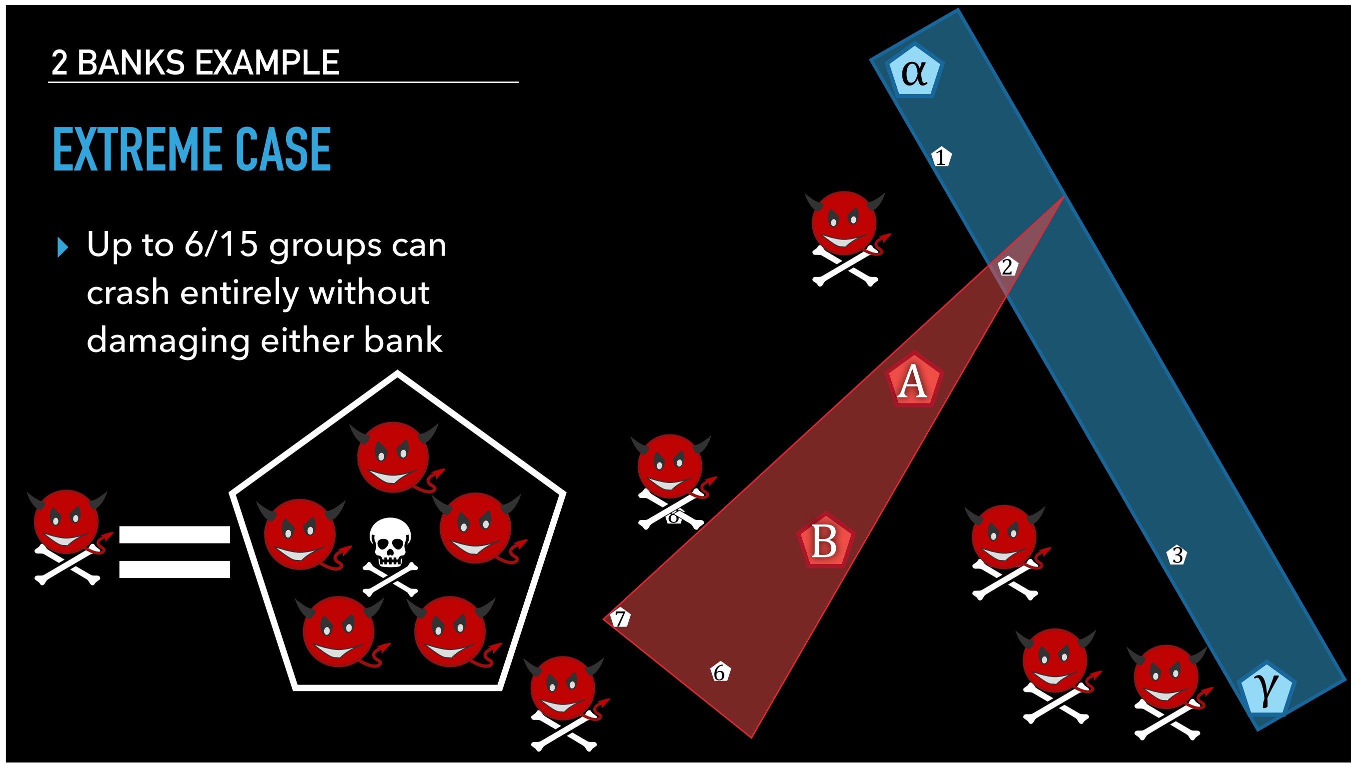
2 BANKS EXAMPLE

EXTREME CASE

- Up to 6/15 groups can crash entirely without damaging either bank

There extreme case that I find amusing.

Recall that up to 6 out of 15 groups can crash without damaging either bank.

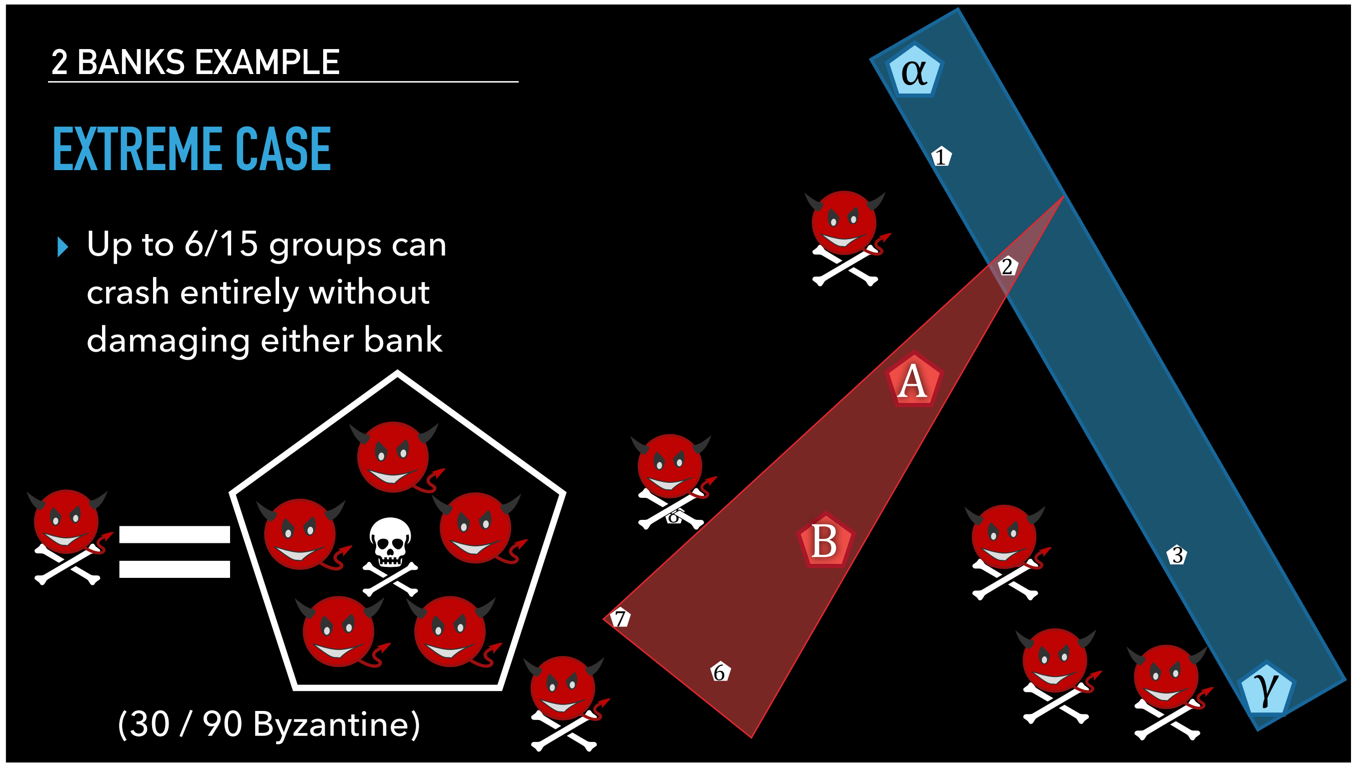


2 BANKS EXAMPLE

EXTREME CASE

- Up to 6/15 groups can crash entirely without damaging either bank

Well, suppose every group has 6 participants,
and instead of just crashing, those 6 groups each feature 5 Byzantine participants and one crashed one.



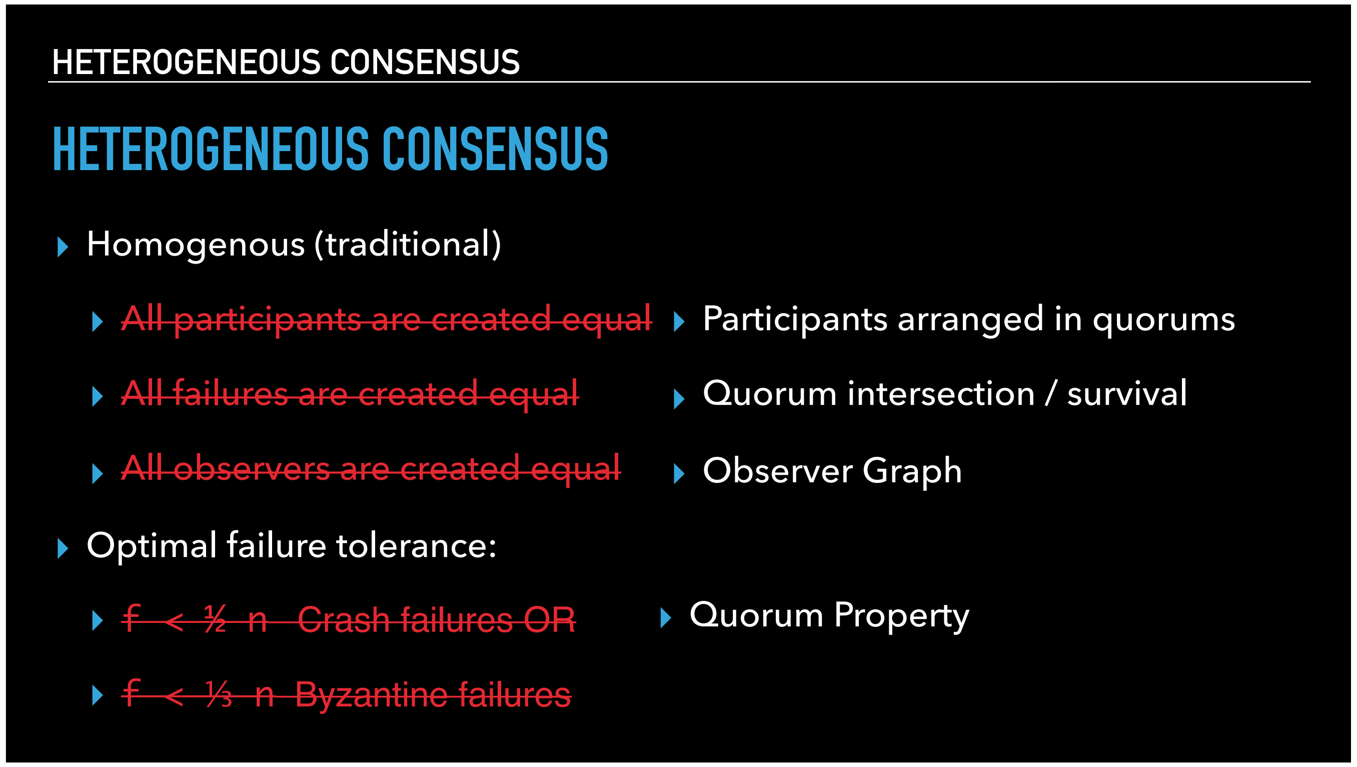
2 BANKS EXAMPLE

EXTREME CASE

- Up to 6/15 groups can crash entirely without damaging either bank

(30 / 90 Byzantine)

Now we have 90 participants, and 30, fully a third of them, are Byzantine, and yet the Banks go on undamaged.
That's just not possible in a Homogeneous Consensus.


HETEROGENEOUS CONSENSUS

HETEROGENEOUS CONSENSUS

- Homogenous (traditional)
  - ~~All participants are created equal~~
  - ~~All failures are created equal~~
  - ~~All observers are created equal~~
- Participants arranged in quorums
- Quorum intersection / survival
- Observer Graph
- Optimal failure tolerance:
  - ~~$f < ½ n$ Crash failures OR~~
  - ~~$f < ⅓ n$ Byzantine failures~~
- Quorum Property

So this is our Heterogeneous Consensus.
Not all participants are created equal: they're arranged in quorums.
Not all failures are created equal: we have Byzantine and Crash failures as long as they obey the Quorum Property.
Not all observers are created equal: we have an observer graph.

Now, in the Homogeneous case, our failure tolerance reduces to the known optimal bounds, but our Quorum Property strictly generalizes that.

RELATED WORK

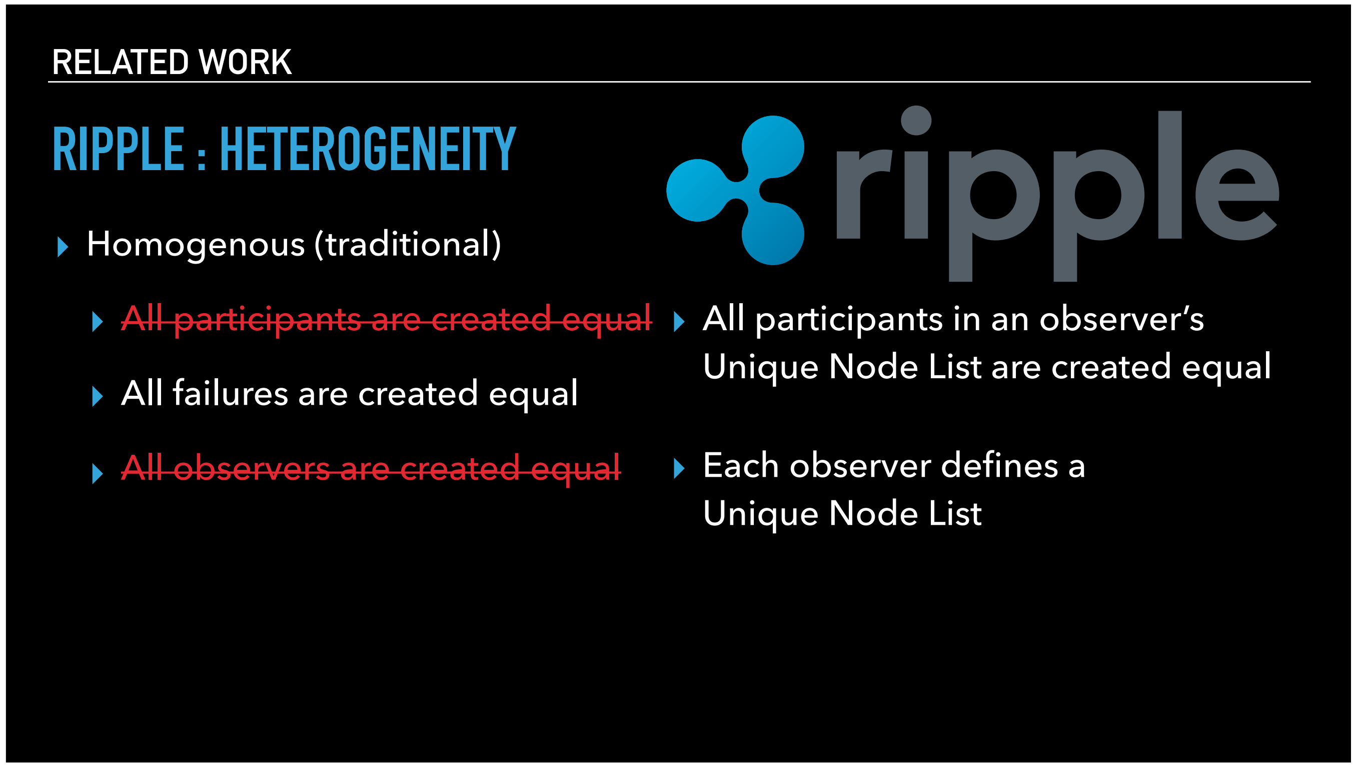
RELATED WORK

ripple

stellar

Now there are other Heterogeneous Consensus projects out there, although they don’t call themselves that.
As we’ve already mentioned, Paxos can have Heterogeneous Participants and Heterogeneous Failures, but not Heterogeneous Observers.

But Stellar and Ripple are notable as the other projects with Heterogeneous Observers.


RELATED WORK

RIPPLE : HETEROGENEITY

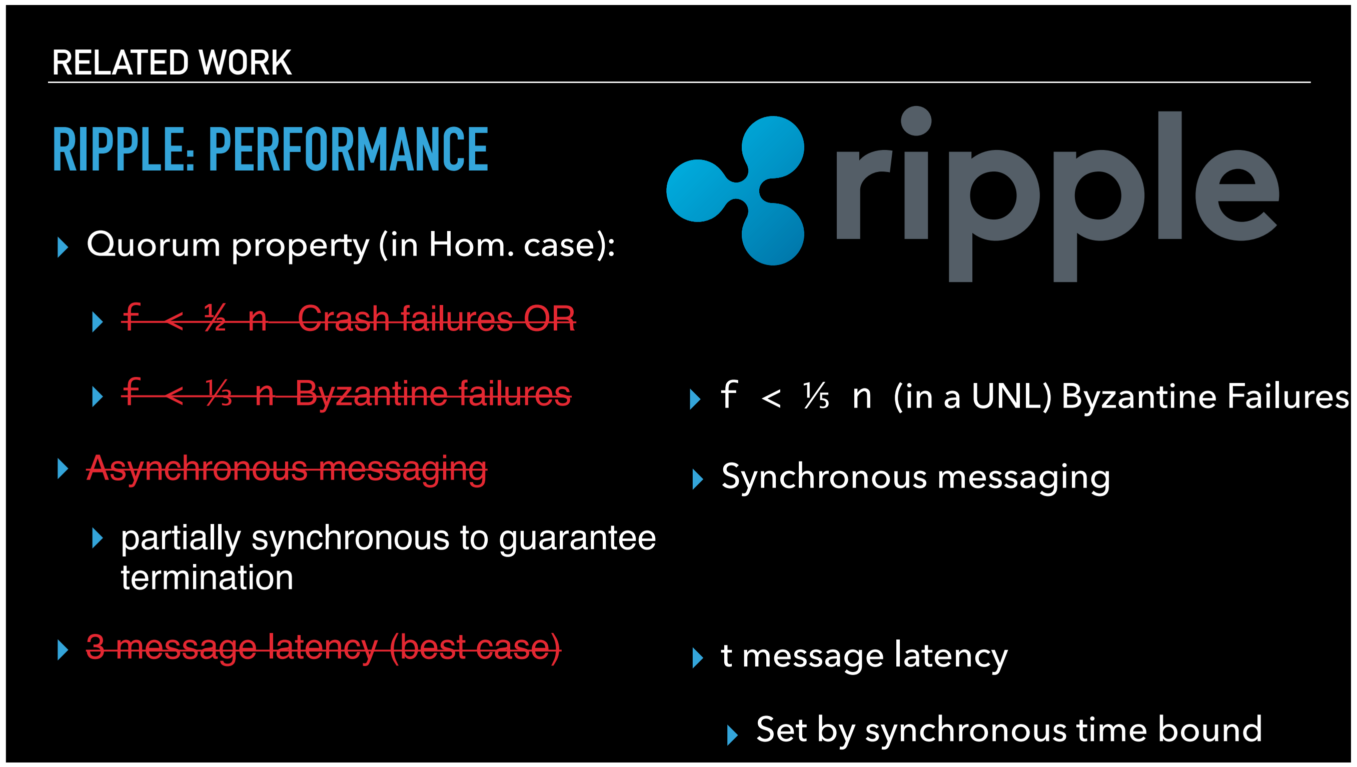
- Homogenous (traditional)
  - ~~All participants are created equal~~
  - All failures are created equal
  - ~~All observers are created equal~~

- All participants in an observer's Unique Node List are created equal
- Each observer defines a Unique Node List

I'll start with Ripple.

Ripple allows each participant to make their own "Unique Node List," featuring a subset of the participants.
From that Observer's perspective, all participants on its Unique Node List are created equal, and the other ones don't matter.

It seems to me that our Observer graph is strictly more general.



RELATED WORK

RIPPLE: PERFORMANCE

- Quorum property (in Hom. case):
  - ~~f < ½ n Crash failures OR~~
  - ~~f < ⅓ n Byzantine failures~~
- ~~Asynchronous messaging~~
  - partially synchronous to guarantee termination
- ~~3 message latency (best case)~~

- f < ⅕ n (in a UNL) Byzantine Failures
- Synchronous messaging
- t message latency
  - Set by synchronous time bound

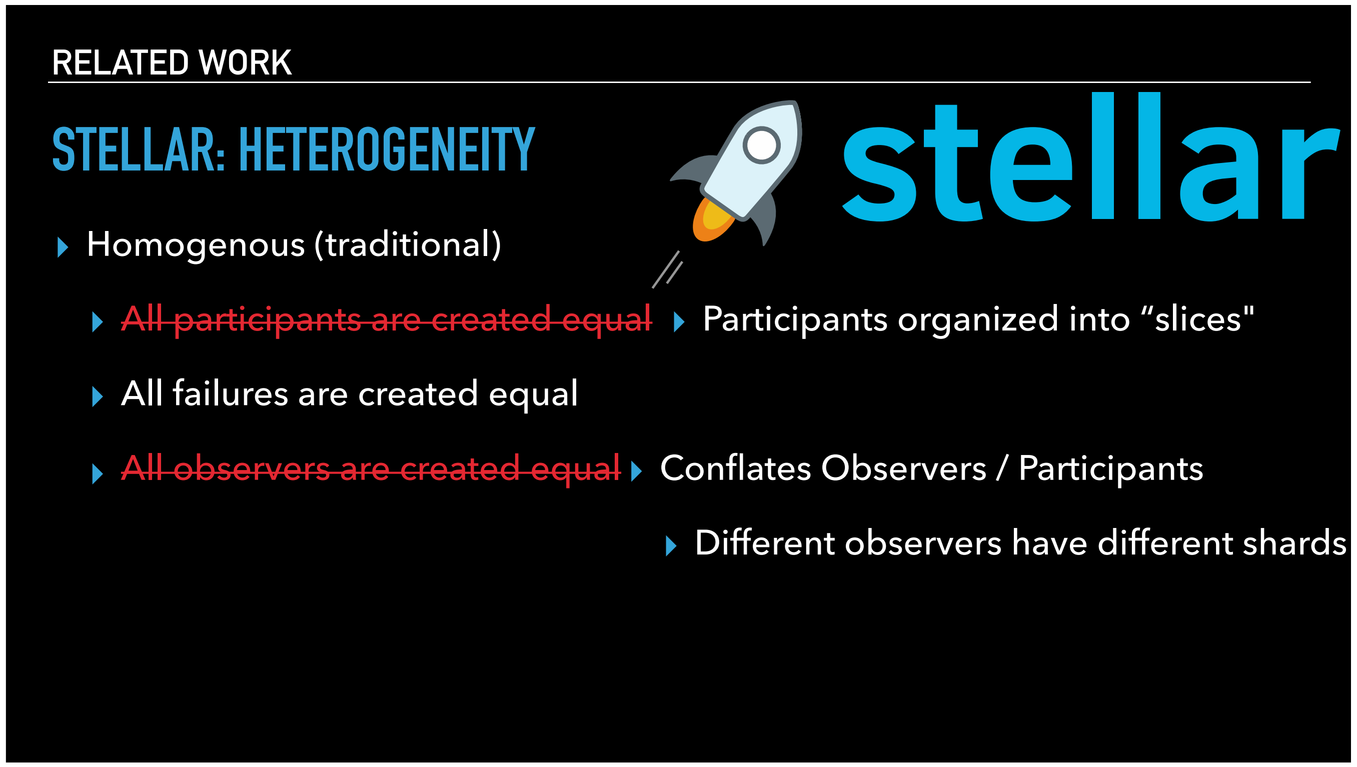
On to performance.

It’s easiest to compare Ripple’s failure tolerance to the Heterogeneous case, in particular because it requires that 80% of the participants in a Unique Node List are honest.
This contrasts with the 67% in a homogeneous Byzantine Consensus.

Ripple also requires large overlaps in the Unique Node Lists of all the observers.

They rely on some different assumptions, in particular Synchronous Messaging, which means they terminate in exactly t message sends, where t is set by a synchronous time bound.

That’s not as good in the best case as the 3 message send best case we inherited from Paxos, but potentially better in the worst case.



RELATED WORK

STELLAR: HETEROGENEITY

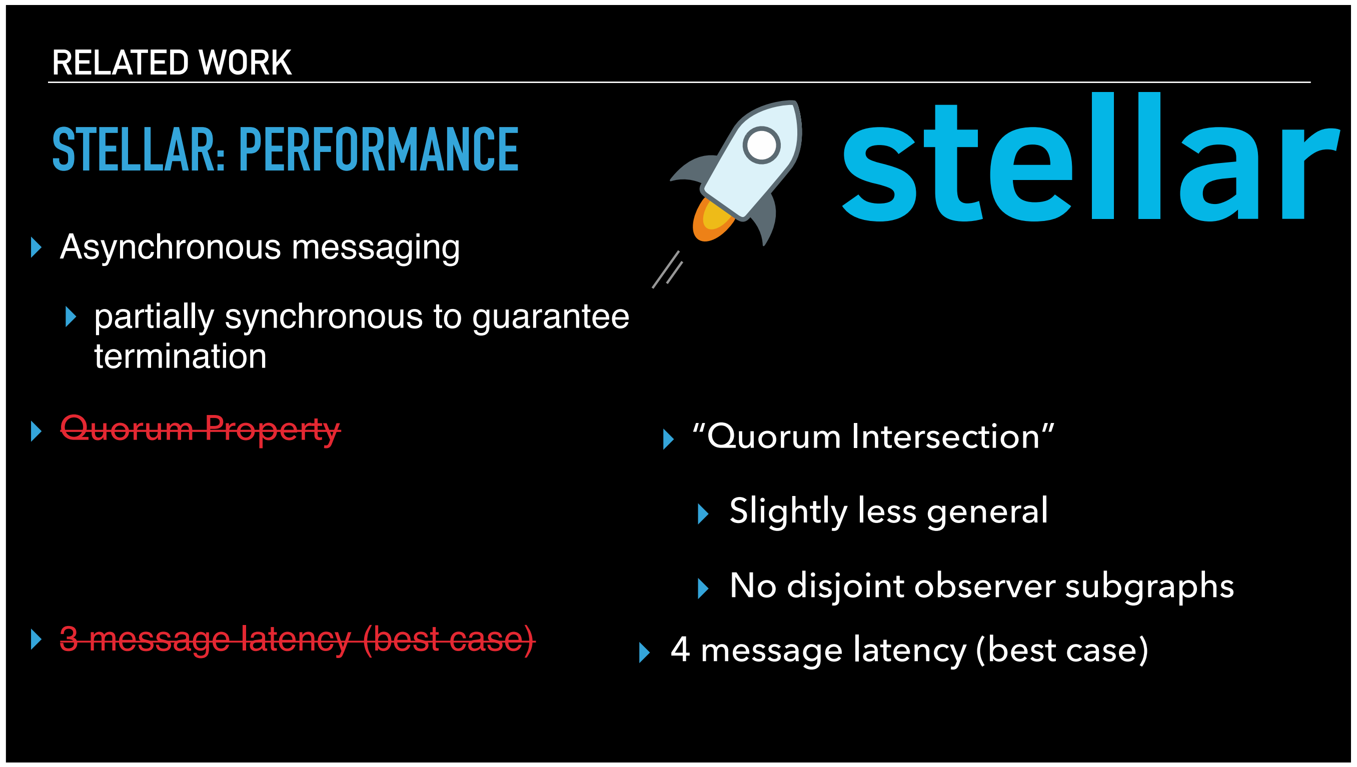
stellar

- Homogenous (traditional)
  - ~~All participants are created equal~~
  - All failures are created equal
  - ~~All observers are created equal~~
- Participants organized into "slices"
- Conflates Observers / Participants
  - Different observers have different shards

Stellar is perhaps a better comparison.

Not all participants are created equal.
Each observer chooses "Slices" of participants such that they want to agree with at least one shard.

This conflates the idea of Participants and Observers, but certainly we can say that not all observers are created equal, either.



RELATED WORK

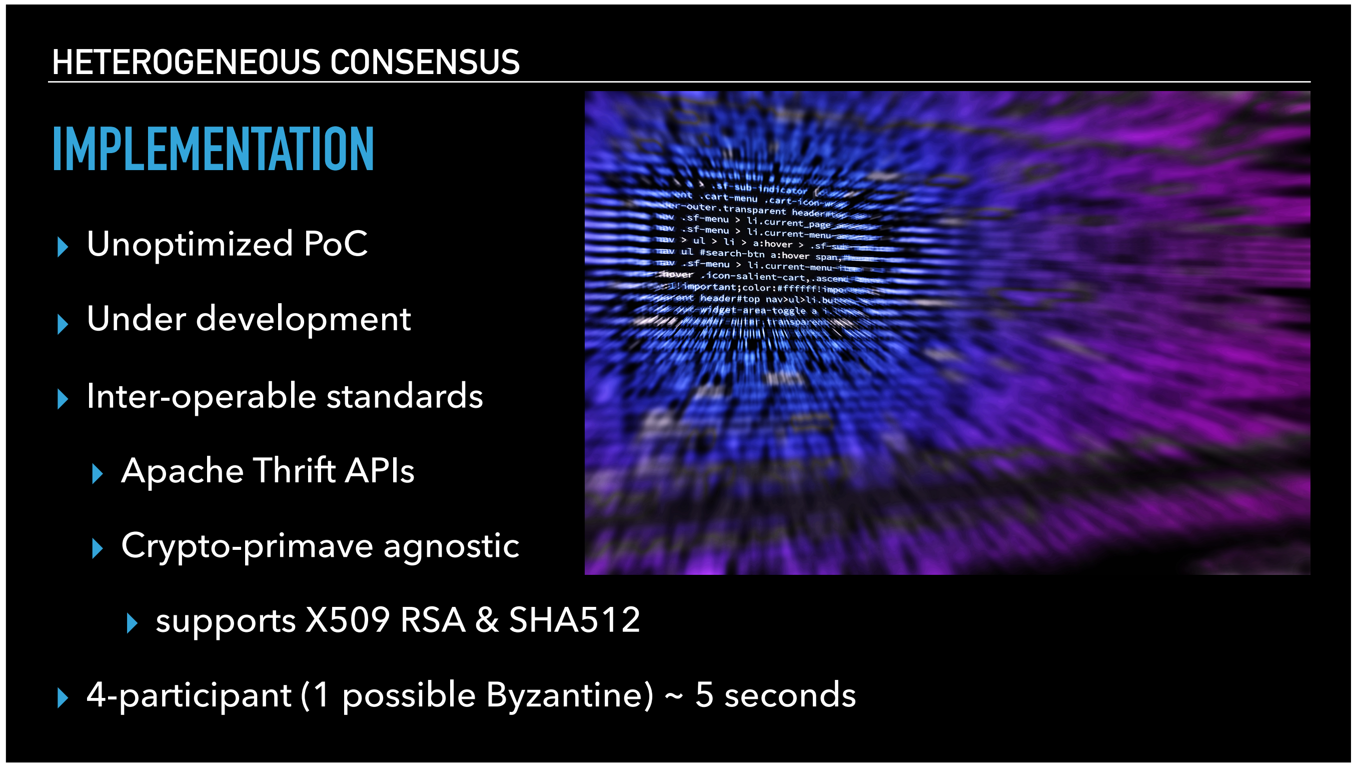
STELLAR: PERFORMANCE

stellar

- Asynchronous messaging
  - partially synchronous to guarantee termination
- ~~Quorum Property~~
- “Quorum Intersection”
  - Slightly less general
  - No disjoint observer subgraphs
- ~~3 message latency (best case)~~
- 4 message latency (best case)

We both have the same synchrony assumptions, and when it comes to the Quorum property,
we have a very similar quorum requirement.
Theirs is slightly less general: either “Quorum Intersection” holds for the whole system, and you get guarantees, or it doesn’t, and you don’t.
We have the possibility of making guarantees for subgraphs of observers who don’t agree under certain failures.
That’s not to say that Stellar can’t prove those guarantees, they just haven’t, that we know of.

Finally, we’re slightly faster than Stellar.
it takes Stellar 4 message sends in the best case to reach consensus, while we inherited a 3-message-send best case from Paxos.



HETEROGENEOUS CONSENSUS

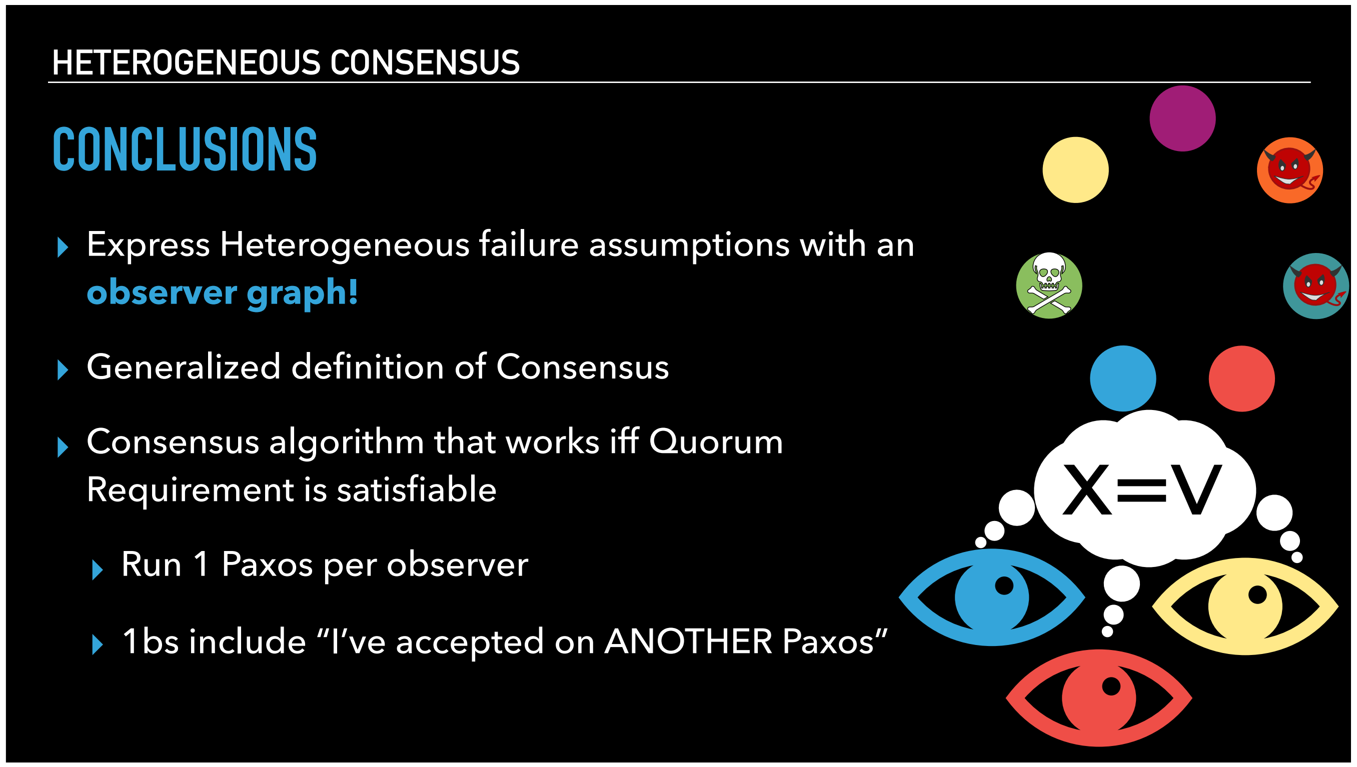
IMPLEMENTATION

- Unoptimized PoC
- Under development
- Inter-operable standards
  - Apache Thrift APIs
  - Crypto-primave agnostic
    - supports X509 RSA & SHA512
- 4-participant (1 possible Byzantine) ~ 5 seconds

We do have an implementation.
It works, but it's very much under development. I wouldn't even call it Alpha at this time.
It's an unoptimized proof-of-concept.
It's never going to be as fast as serious industrial Consensus implementations.
To give you an idea of how "unoptimized," when we express cryptographic identities in the system, its easiest just to use the whole certificate as an identifier, which means that whole certificates are being sent over the wire repeatedly with each message.

We built it with language-independent messaging using Apache thrift, and it's polymorphic across crypto primitives, although right now it only supports X509 with RSA, and SHA512.

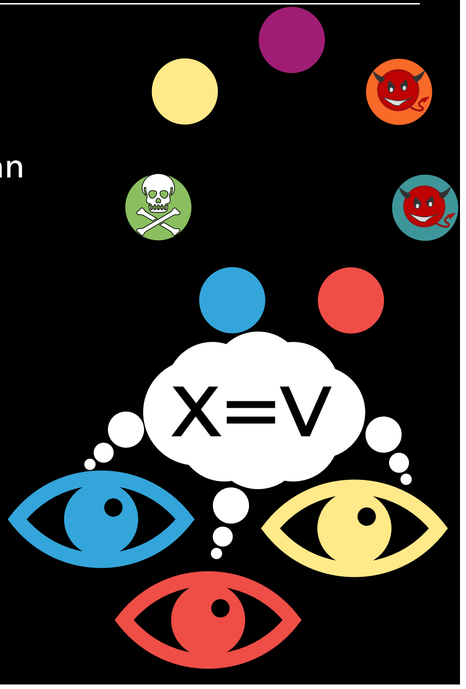
A 4 participant consensus, which tolerates 1 Byzantine failure, takes about 5 seconds to run in the best case right now, but we think that can be brought much, much lower with a more optimized implementation.



HETEROGENEOUS CONSENSUS

CONCLUSIONS

- Express Heterogeneous failure assumptions with an **observer graph!**
- Generalized definition of Consensus
- Consensus algorithm that works iff Quorum Requirement is satisfiable
  - Run 1 Paxos per observer
  - 1bs include "I've accepted on ANOTHER Paxos"



So, we built a Heterogeneous Consensus!

We formalized a definition for what it *means* to even have a heterogeneous Consensus.
We express Heterogeneous failure assumptions with an observer graph,
and we have a quorum requirement that tells us when consensus is possible.

Our consensus, put very briefly, runs 1 pacts per observer, but where 1bs include "I've accepted on ANOTHER Paxos."

HETEROGENEOUS CONSENSUS

https://IsaacSheff.com/hetcons

Isaac Sheff
isheff@cs.cornell.edu

Andrew C. Myers
andru@cs.cornell.edu

Robbert van Renesse
rvr@cs.cornell.edu

And that's it!

You can find links to these slides, implementation code, and maybe future developments at this URL.

Any questions?