Implementing State Machine Replication

- Implement a sequence of separate instances of consensus, where the value chosen by the i^{th} instance is the i^{th} message in the sequence.
- Each server assumes all roles in each instance of the algorithm.
- Assume that the set of servers is fixed

The role of the leader

- In normal operation, elect a single server to be a leader. The leader acts as a distinguished proposer in all instances of the consensus algorithm.
  - Clients send commands to the leader, which decides where in the sequence each command should appear.
  - If the leader, for example, decides that a client command is the k^{th} command, it tries to have the command chosen as the value in the k^{th} instance of consensus.

What if a new λ is elected?

- Since λ serves also as a replica in all instances of consensus, it should know most of the commands that have already been chosen. For example, it might know commands for slots 1-10, 13, and 15.
  - It executes phase 1 for slots 11, 12, and 14 and of all slots 16 and larger.
  - λ may find that some value was already accepted for slots 14 and 16 and that slots 11, 12 and all slots after 16 have accepted no command.
  - λ then executes phase 2 of 14 and 16, using the value with the highest ballot it retrieved for those slots

What if II: Stop-gap measures

- All replicas now can execute commands 1-10, but not 13-16 because 11 and 12 haven't yet been chosen.
- λ can either take the next two commands it receives by clients to be commands 11 and 12, or can propose immediately that 11 and 12 be no-op commands.
  - this is what happens on "Olive Day"!
- λ runs phase 2 of consensus for slots 11 and 12.
- Once consensus is achieved, all replicas can execute all commands through 16.

To infinity, and beyond

- λ can efficiently execute phase 1 for infinitely many instances of consensus! (e.g. command 16 and higher)
  - λ just sends a message with a sufficiently high ballot number for all instances
  - An acceptor replies non trivially only for instances for which it has already accepted a value

Paxos and FLP

- Paxos is always safe–despite asynchrony
- Once a leader is elected, Paxos is live.
-  "Ciao ciao" FLP?
  - To be live, Paxos requires a single leader
  - "Leader election" is impossible in an asynchronous system (gotcha!)
- Given FLP, Paxos is the next best thing: always safe, and live during periods of synchrony

A Lower Bound

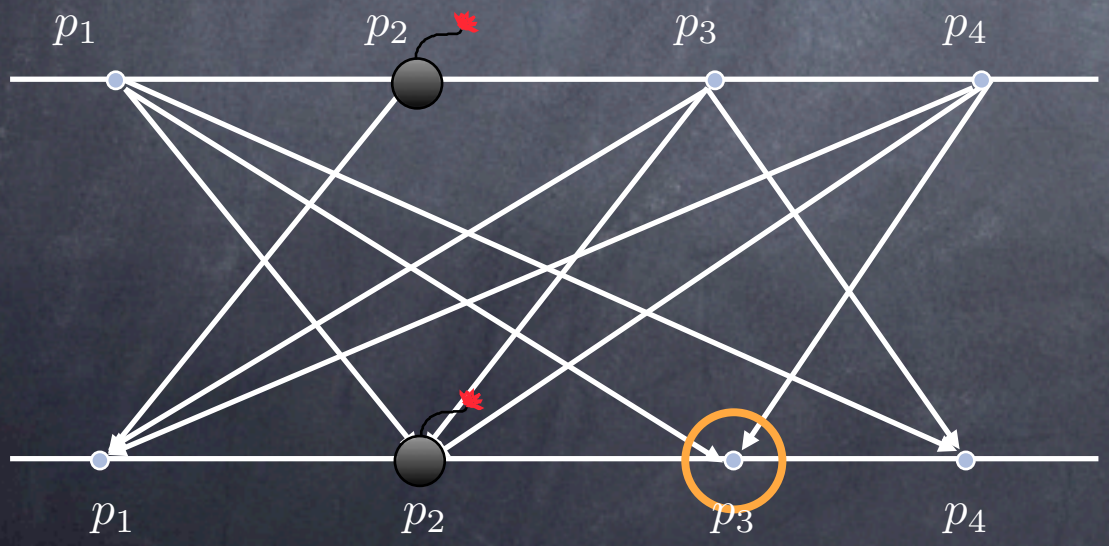
Theorem

There is no algorithm that solves the consensus problem in fewer than $f+1$ rounds in the presence of f crash failures, if $n \geq f+2$

We consider a special case $(f=1)$ to study the proof technique

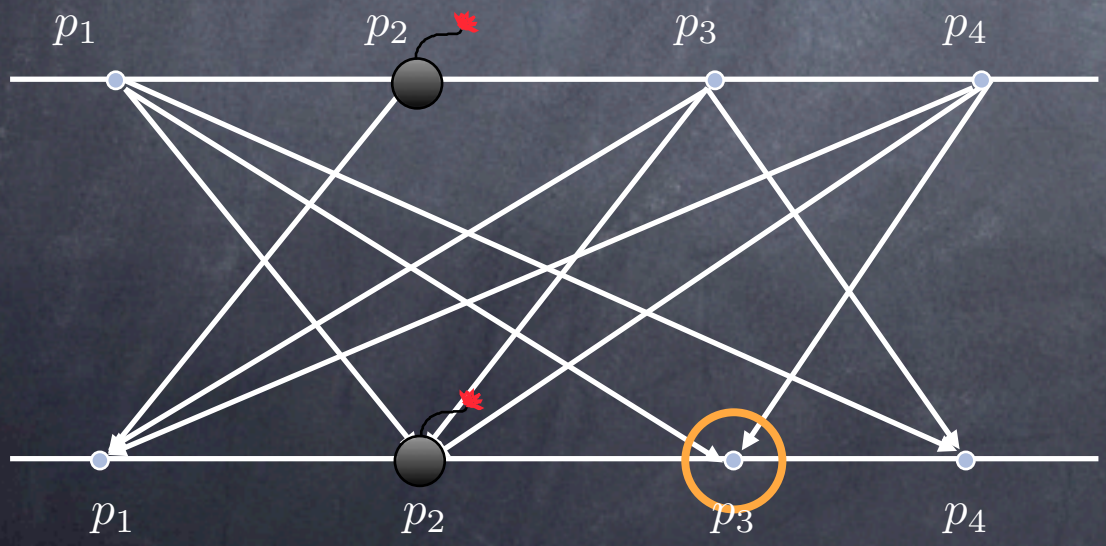
Views

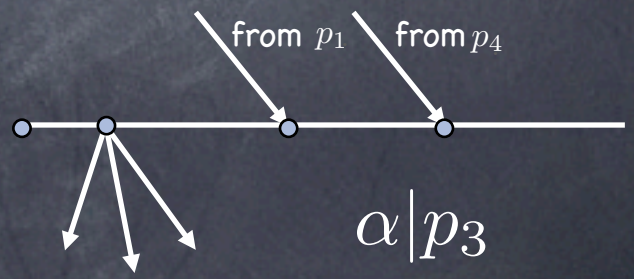
Let α be an execution. The view of process p_i in α, denoted by $\alpha|p_i$, is the subsequence of computation and message receive events that occur in p_i together with the state of p_i in the initial configuration of α



Views

Let α be an execution. The view of process p_i in α, denoted by $\alpha|p_i$, is the subsequence of computation and message receive events that occur in p_i together with the state of p_i in the initial configuration of α





Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2.

α_1 is similar to α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if

$$\alpha_1|p_i = \alpha_2|p_i$$

Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2.
α_1 is similar to α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if

$$\alpha_1|p_i = \alpha_2|p_i$$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then p_i decides the same value in both executions

Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2.
α_1 is similar to α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if

$$\alpha_1|p_i = \alpha_2|p_i$$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then p_i decides the same value in both executions

Lemma If $\alpha_1 \sim_{p_i} \alpha_2$ and p_i is correct, then dec(α_1) = dec(α_2)

Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2.
α_1 is similar to α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if

$$\alpha_1|p_i = \alpha_2|p_i$$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then p_i decides the same value in both executions

Lemma If $\alpha_1 \sim_{p_i} \alpha_2$ and p_i is correct, then dec(α_1) = dec(α_2)

The transitive closure of $\alpha_1 \sim_{p_i} \alpha_2$ is denoted $\alpha_1 \approx \alpha_2$.

We say that $\alpha_1 \approx \alpha_2$ if there exist executions $\beta_1, \beta_2, \ldots, \beta_{k+1}$ such that

$$\alpha_1 = \beta_1 \sim_{p_{i_1}} \beta_2 \sim_{p_{i_2}} \ldots, \sim_{p_{i_k}} \beta_{k+1} = \alpha_2$$

Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2.
α_1 is similar to α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if

$$\alpha_1|p_i = \alpha_2|p_i$$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then p_i decides the same value in both executions

Lemma If $\alpha_1 \sim_{p_i} \alpha_2$ and p_i is correct, then dec(α_1) = dec(α_2)

The transitive closure of $\alpha_1 \sim_{p_i} \alpha_2$ is denoted $\alpha_1 \approx \alpha_2$.

We say that $\alpha_1 \approx \alpha_2$ if there exist executions $\beta_1, \beta_2, \ldots, \beta_{k+1}$ such that

$$\alpha_1 = \beta_1 \sim_{p_{i_1}} \beta_2 \sim_{p_{i_2}} \ldots, \sim_{p_{i_k}} \beta_{k+1} = \alpha_2$$

Lemma If $\alpha_1 \approx \alpha_2$ then dec(α_1) = dec(α_2)

Single-Failure Case

There is no algorithm that solves consensus in fewer than two rounds in the presence of one crash failure, if $n \geq 3$

The Idea

By contradiction

- Consider a one-round execution in which each process proposes 0. What is the decision value?
- Consider another one-round execution in which each process proposes 1. What is the decision value?
- Show that there is a chain of similar executions that relate the two executions.

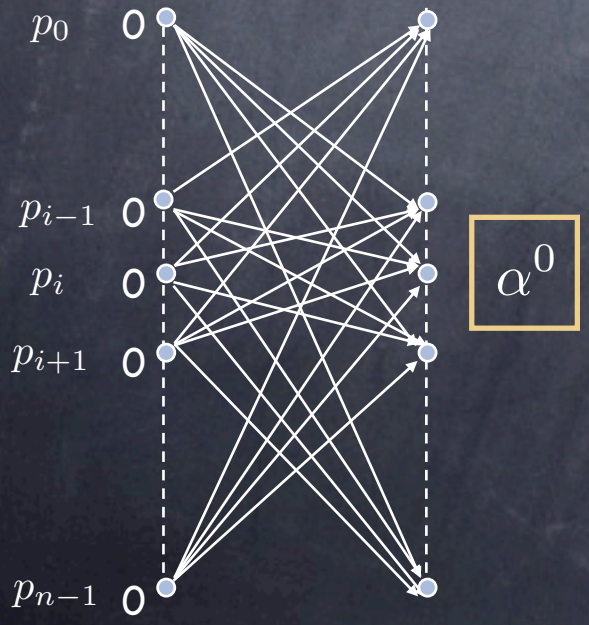
So what?

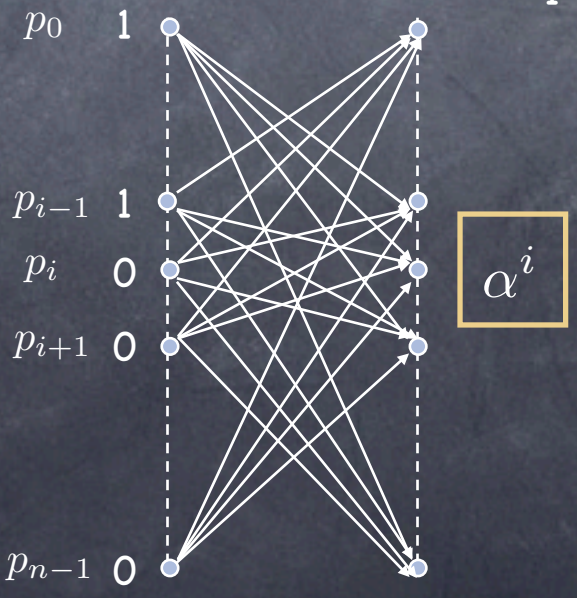
α^is

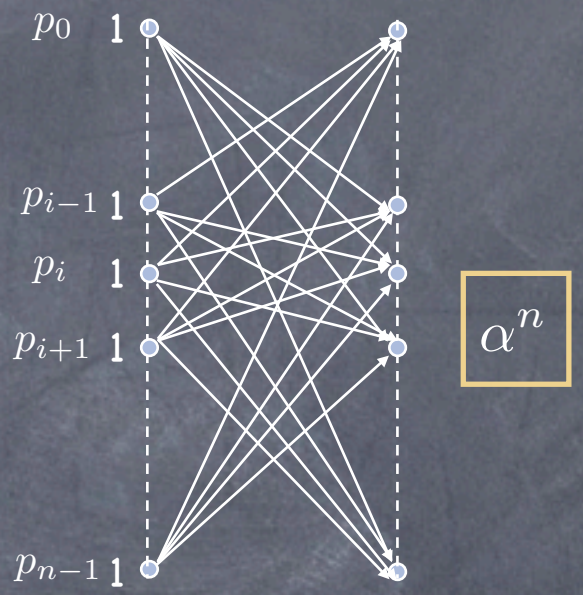
Definition

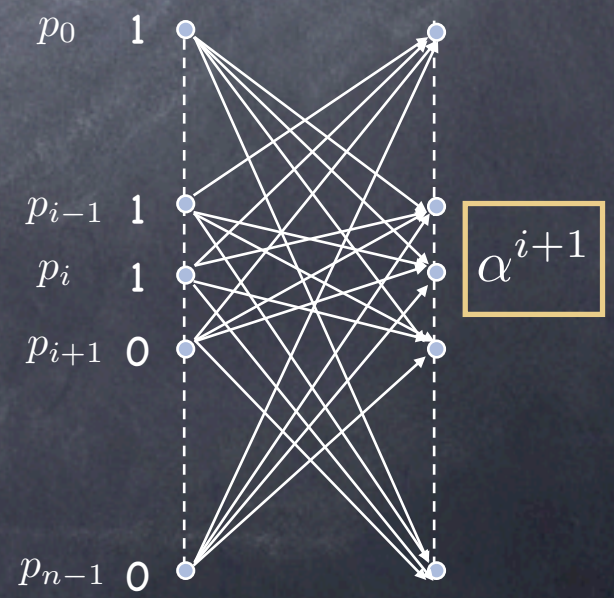
α^i is the execution of the algorithm in which

- no failures occur
- only processes $p_0, \ldots, p_{i-1}$ propose 1







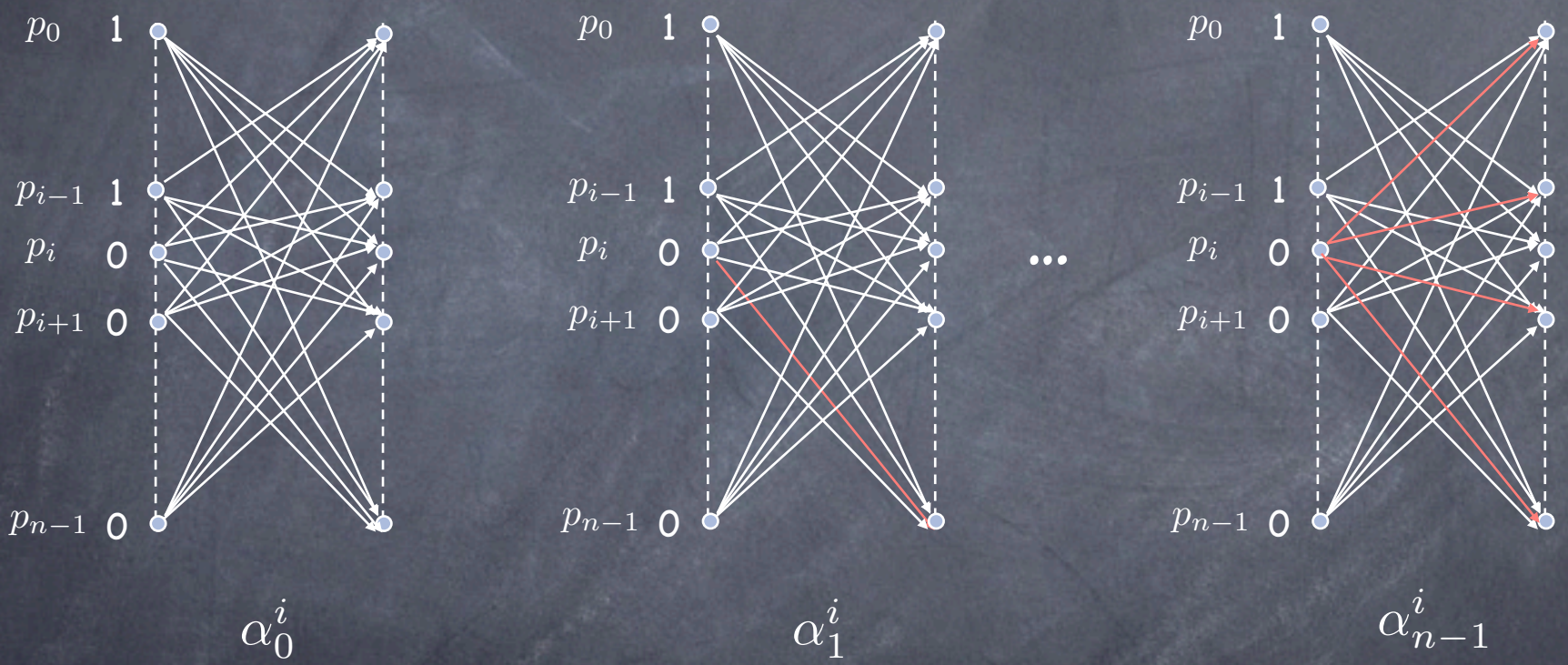


Adjacent α^is are similar!

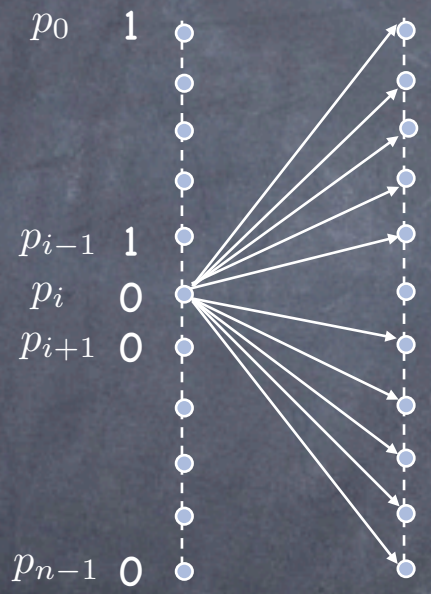
Starting from α^i, we build a set of executions α^i_j where $0 \leq j \leq n-1$as follows:

α^i_j is obtained from α^i after removing the messages that p_i sends to the j-th highest numbered processors (excluding itself)

The executions



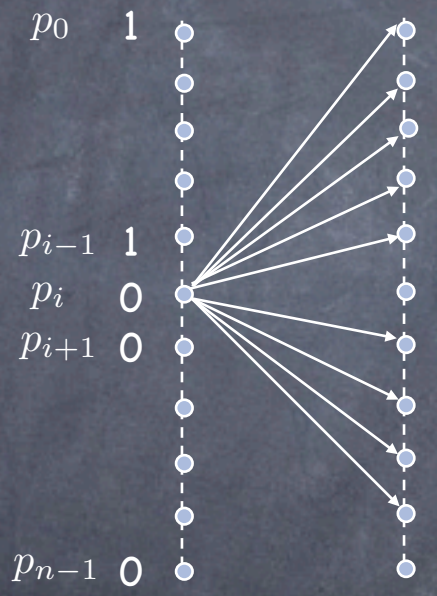
Indistinguishability



α^i

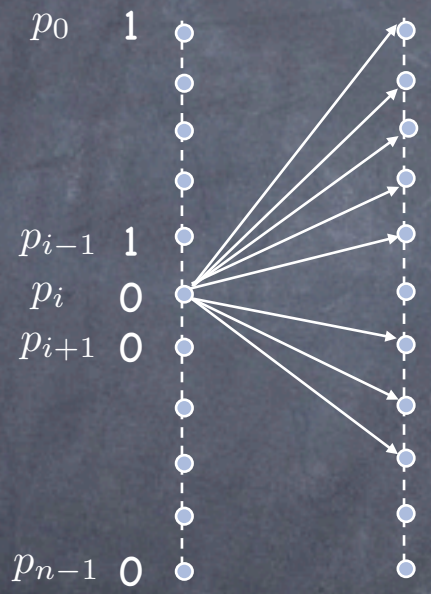
α_0^i

Indistinguishability



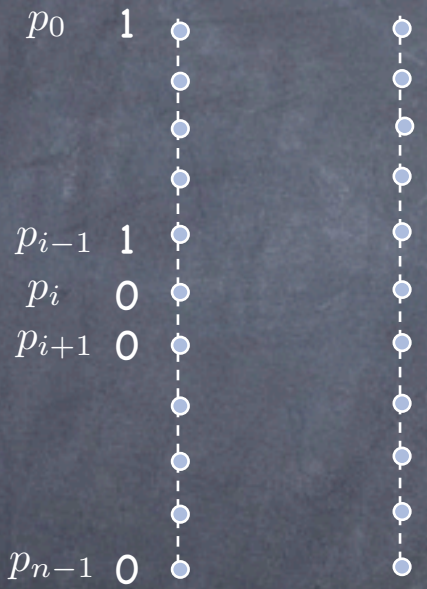
$$\alpha^i \approx \alpha_1^i$$

Indistinguishability



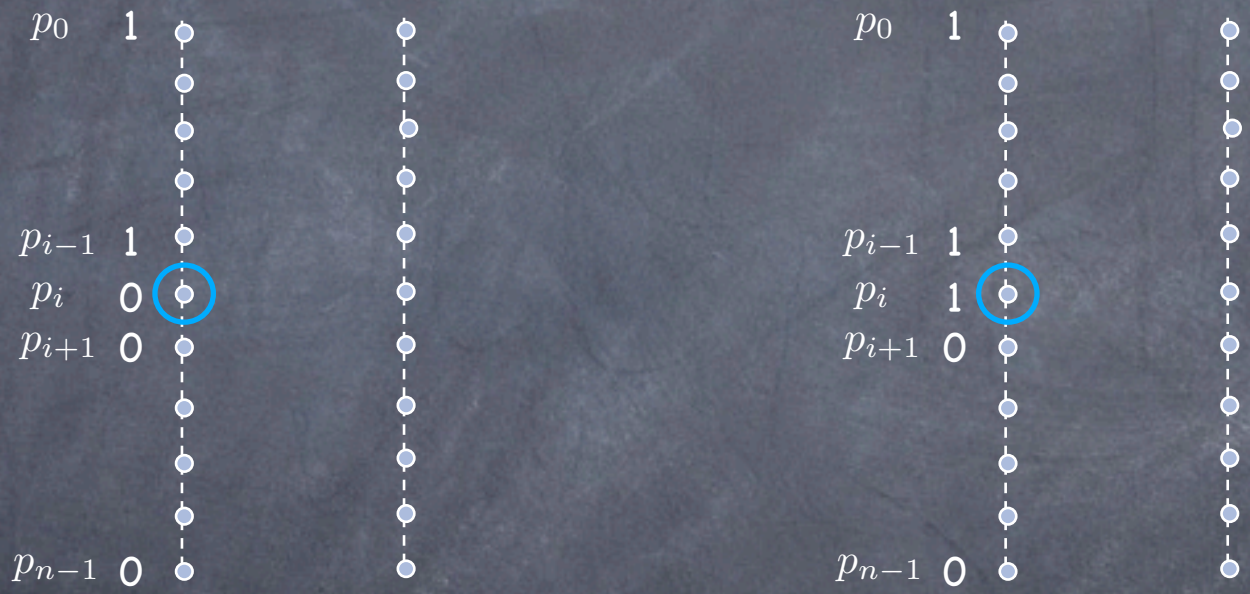
$$\alpha^i \approx \alpha_2^i$$

Indistinguishability



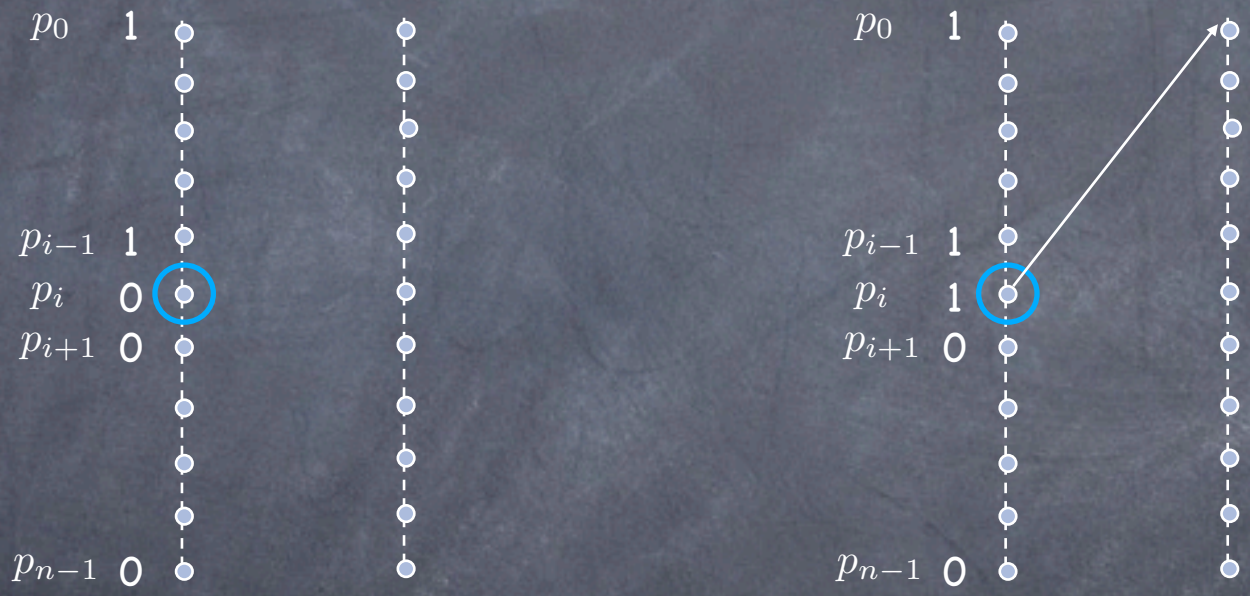
$$\alpha^i \approx \alpha^i_{n-1}$$

Indistinguishability



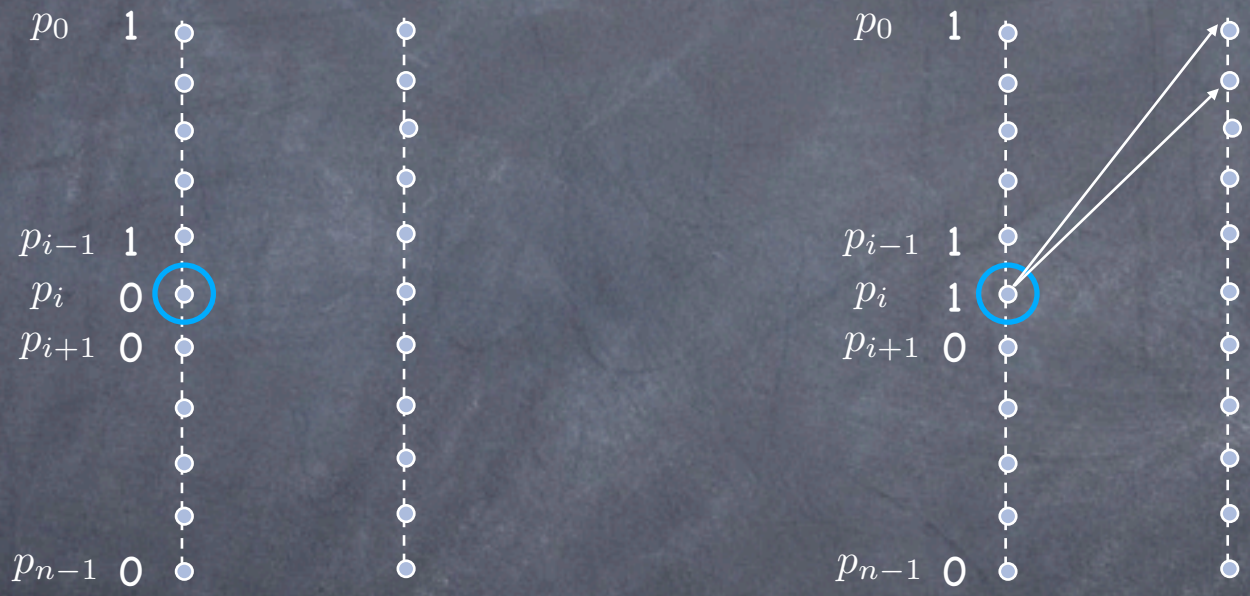
$$\alpha^i \approx \alpha^i_{n-1} \approx \beta^i_{n-1}$$

Indistinguishability



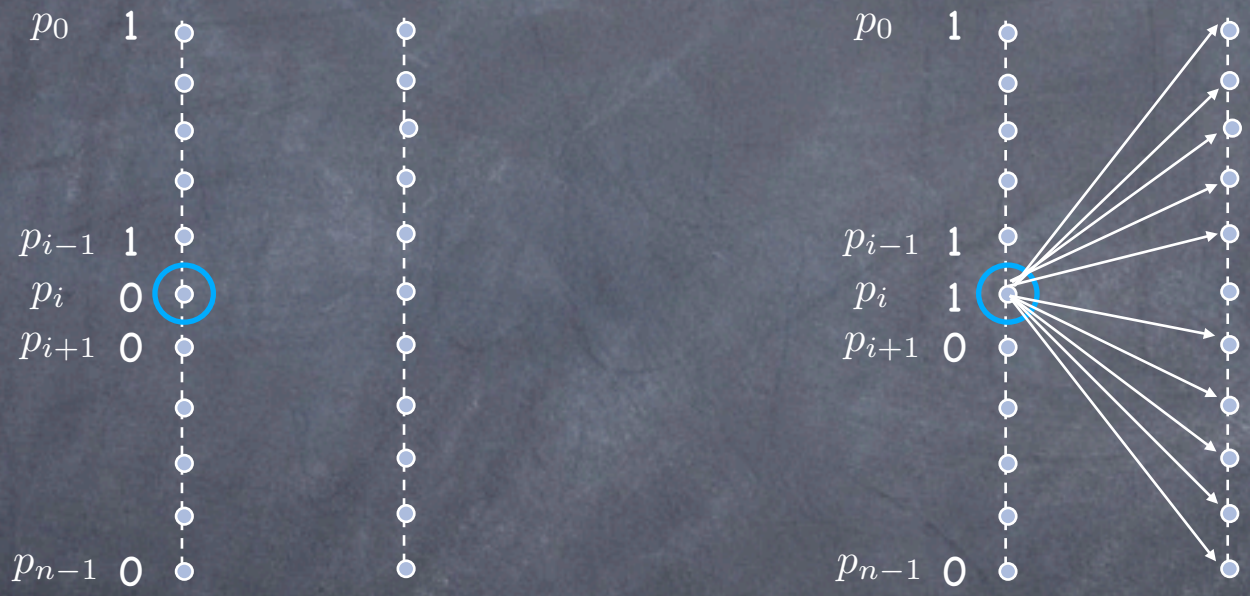
$$\alpha^i \approx \alpha^i_{n-1} \qquad \approx \qquad \beta^i_{n-2}$$

Indistinguishability



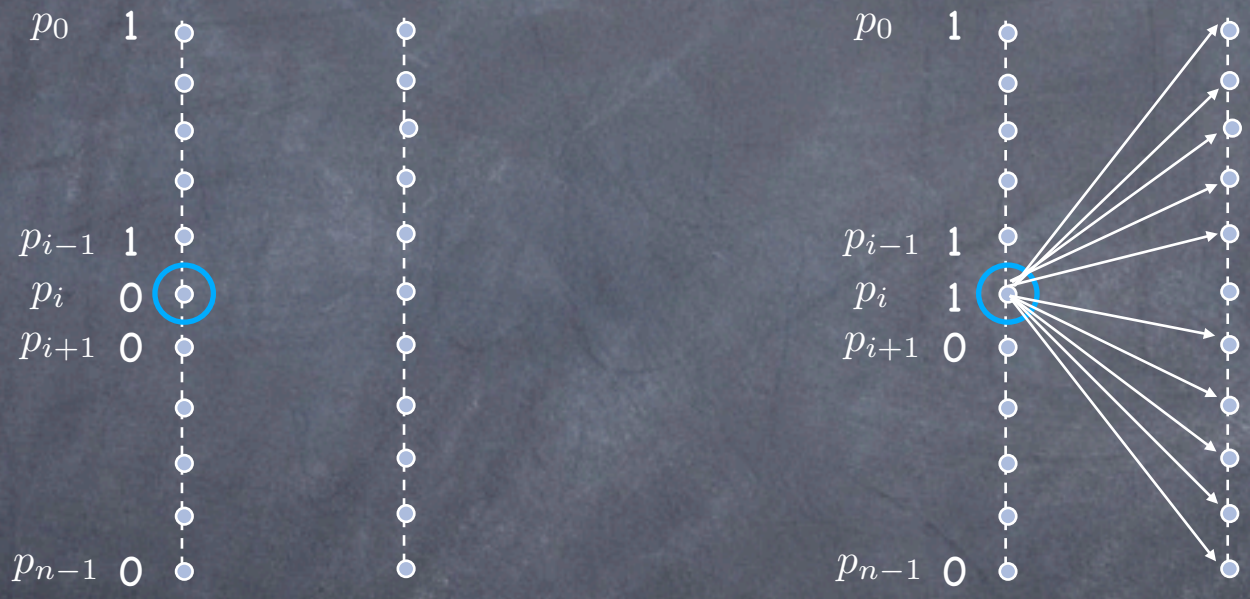
$$\alpha^i \approx \alpha^i_{n-1} \qquad \approx \qquad \beta^i_{n-3}$$

Indistinguishability



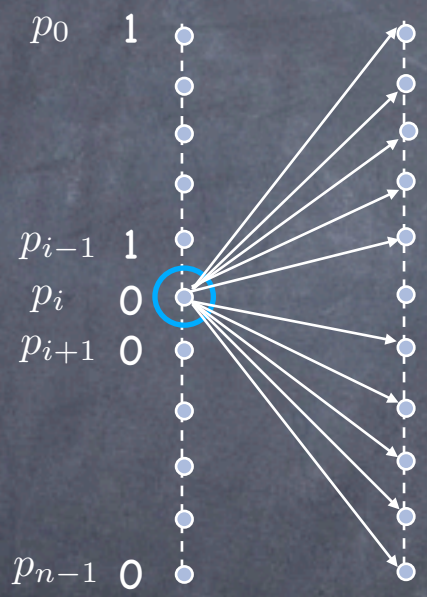
$$\alpha^i \wr \alpha^i_{n-1} \approx \beta^i_0$$

Indistinguishability

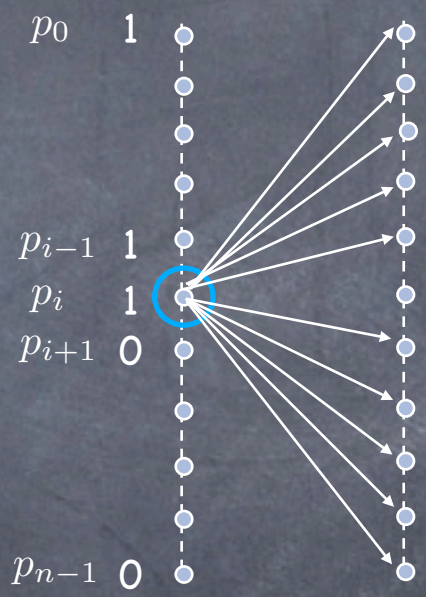


$$\alpha^i \wr \alpha^i_{n-1} \approx \alpha^{i+1} \wr \beta^i_0$$

Indistinguishability

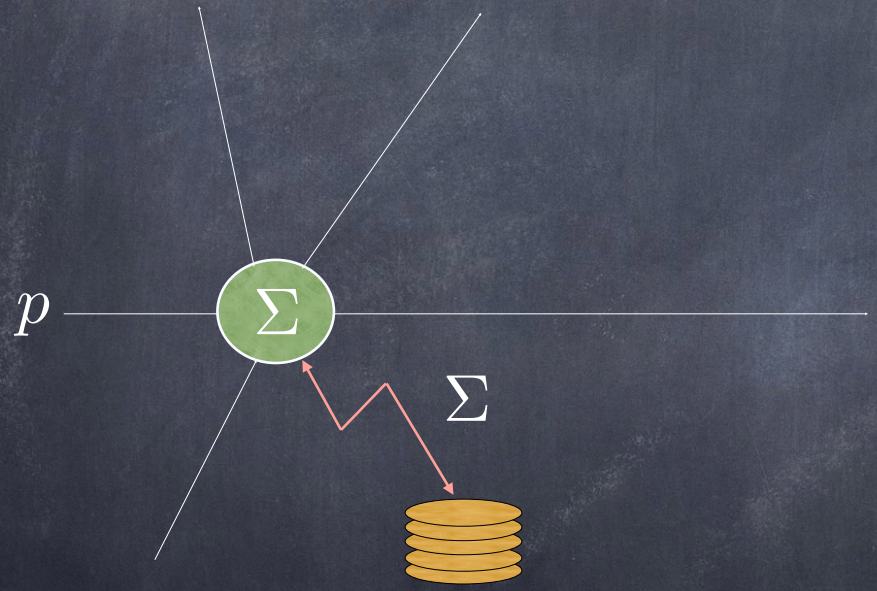


α^i $\approx$ α^{i+1}



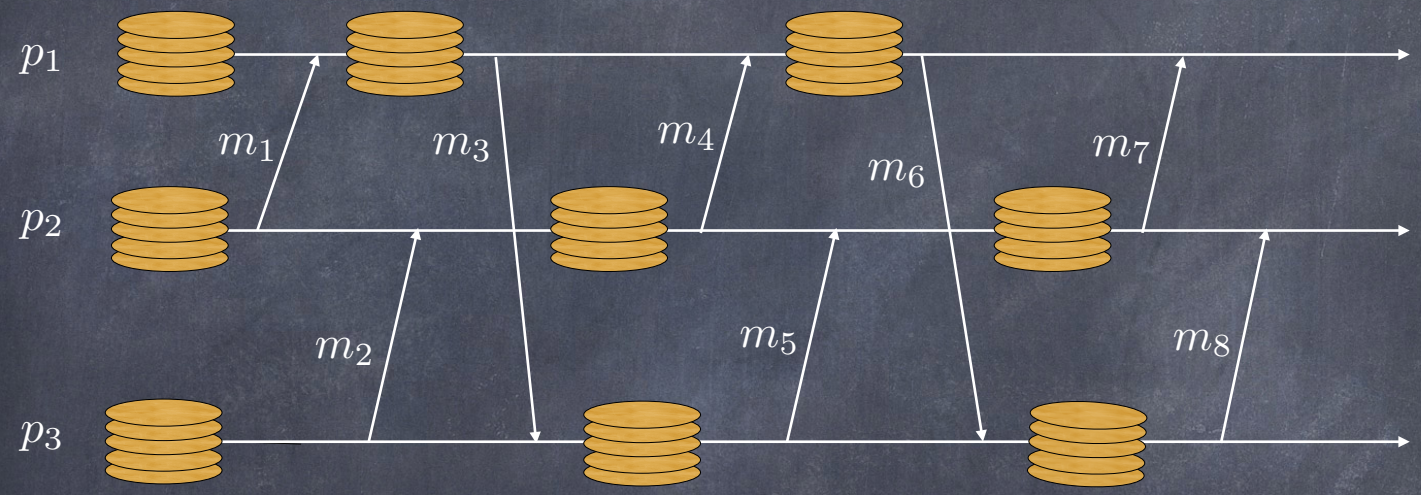
Rollback-Recovery

Uncoordinated Checkpointing

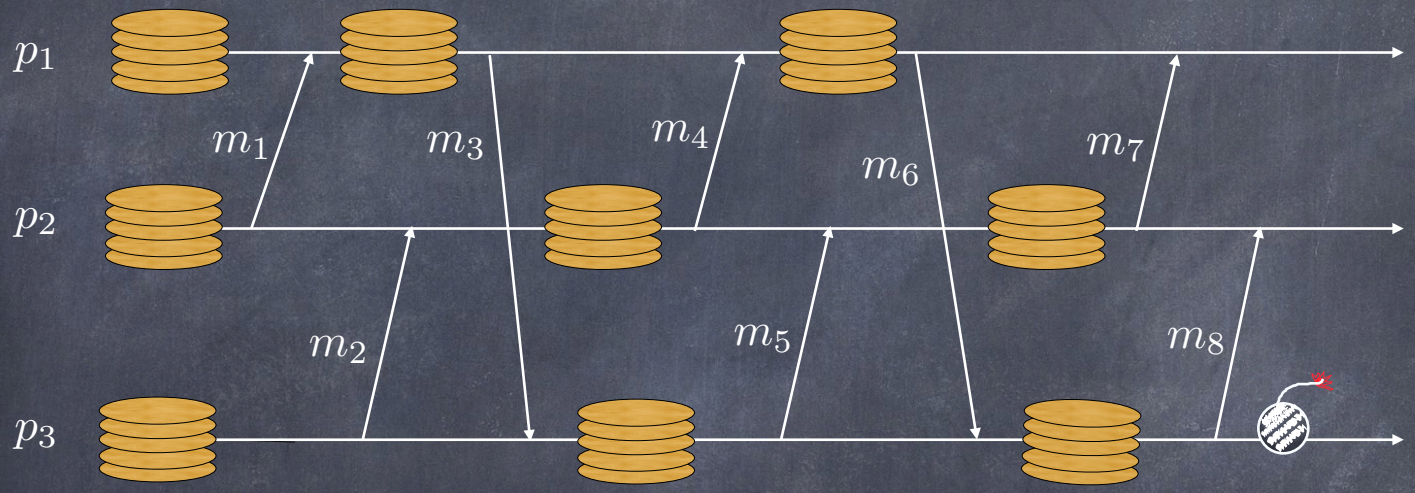


- Easy to understand
- No synchronization overhead
- Flexible
  - can choose when to checkpoint
- To recover from a crash:
  - go back to last checkpoint
  - restart

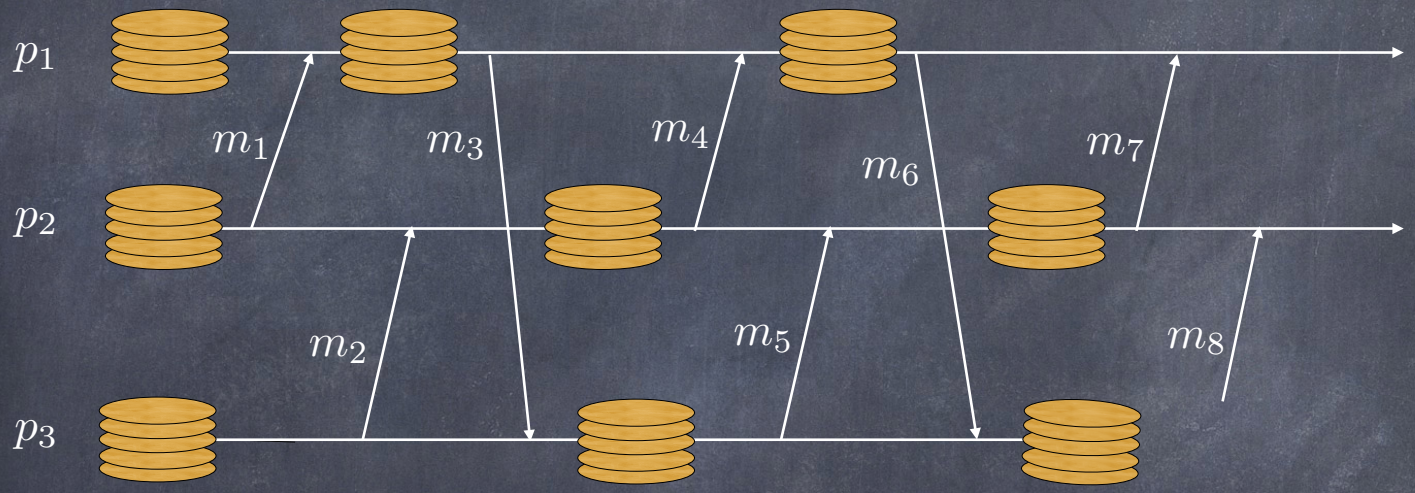
The Domino Effect



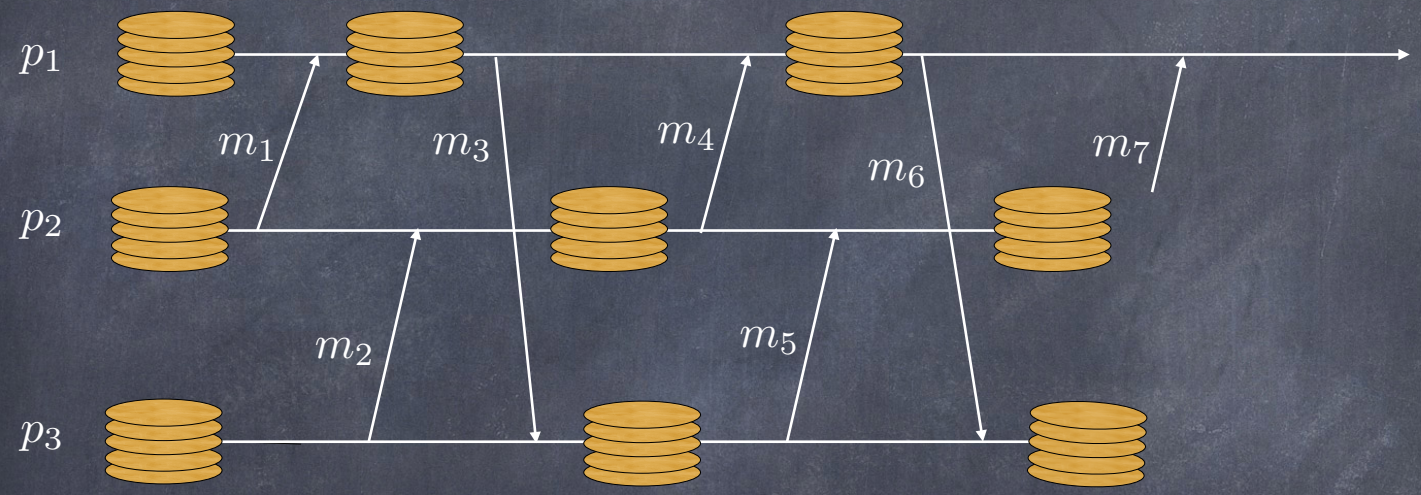
The Domino Effect



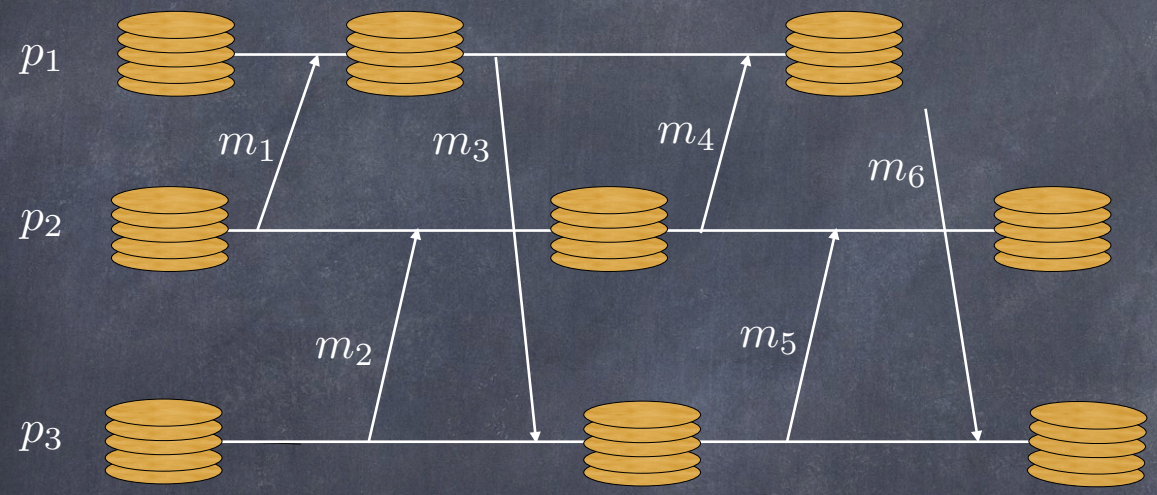
The Domino Effect



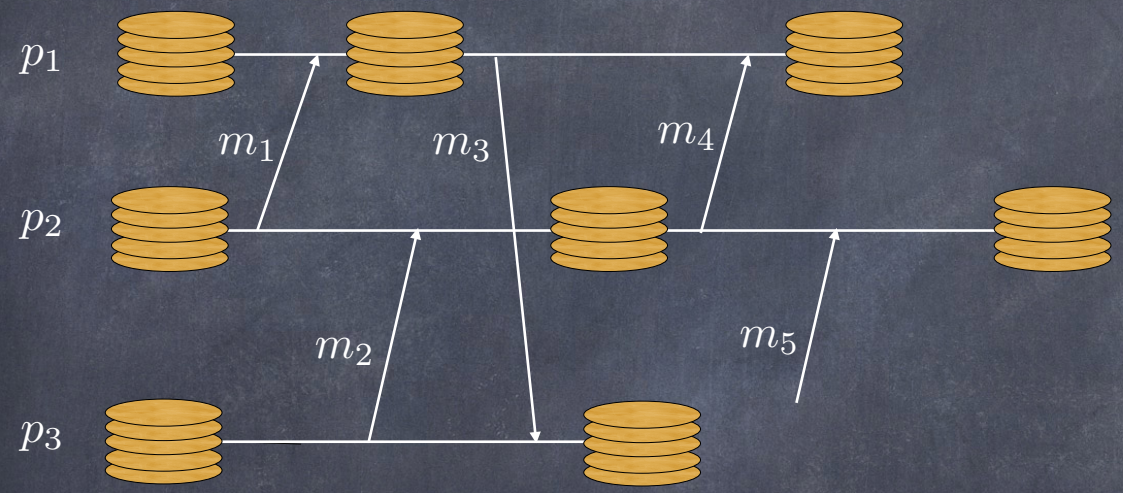
The Domino Effect



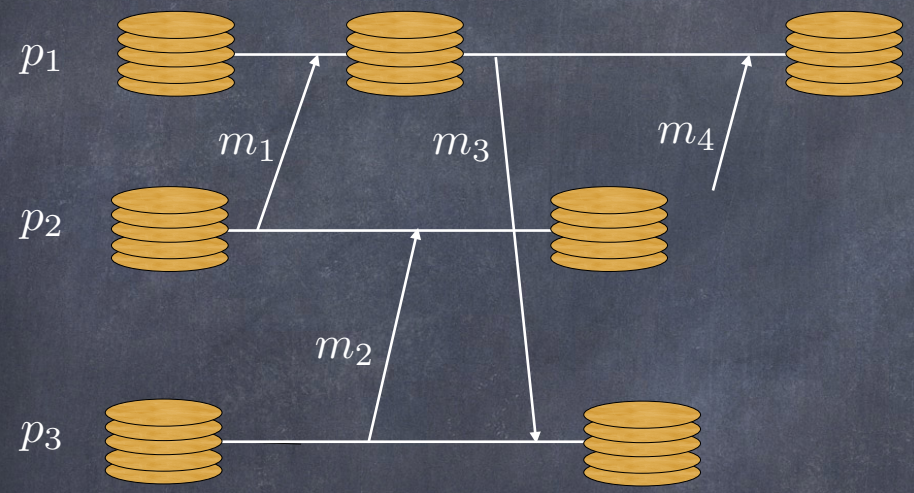
The Domino Effect



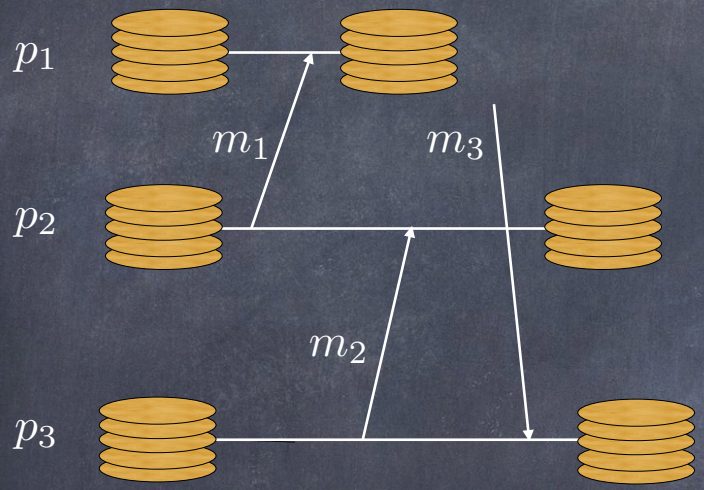
The Domino Effect



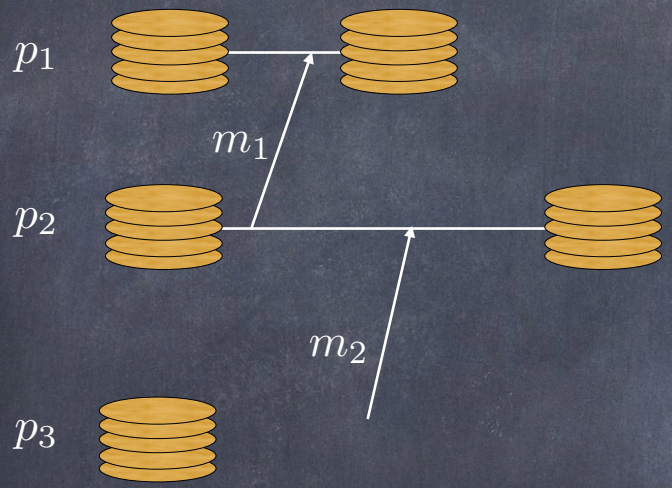
The Domino Effect



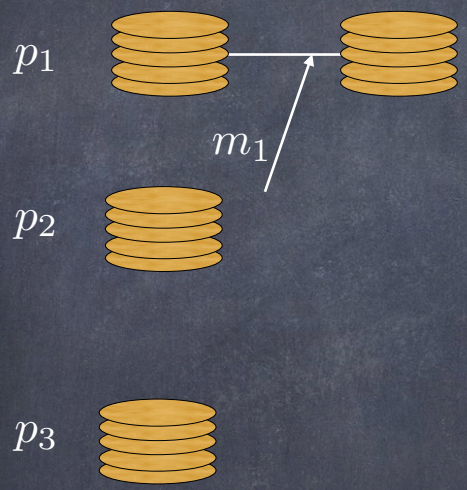
The Domino Effect



The Domino Effect



The Domino Effect



The Domino Effect

p_1 

p_2 

p_3