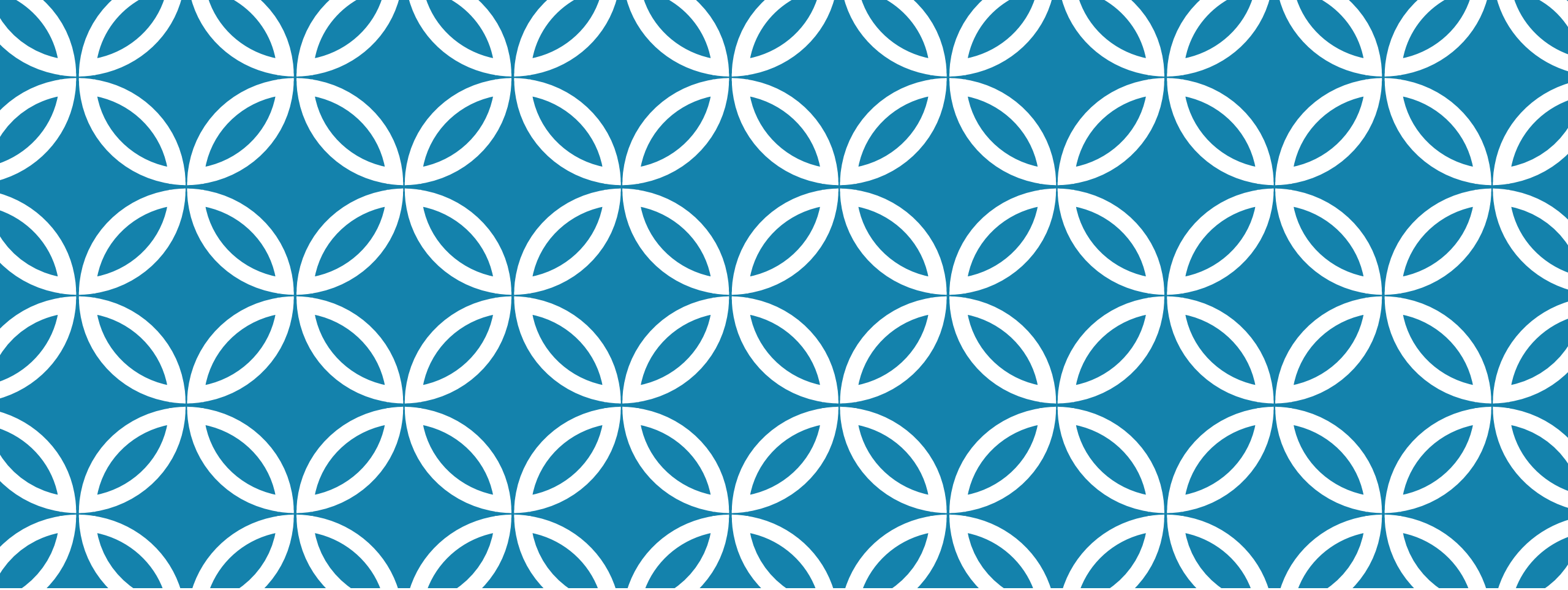


MICROSOFT'S FARM KEY-VALUE STORE

Professor Ken Birman

CS4414 Lecture 23

IDEA MAP FOR TODAY

Modern applications often work with big data

By definition, big data means "you can't fit it on your machine"
Reminder: Shared storage servers accessed over a network.

Reminder: MemCacheD (Lecture 22)

Concept: Transactions. Applying this concept to a key-value store.

RDMA hardware accelerator for TCP and remote memory access. How FaRM leveraged this.

Ken spent a sabbatical at MSRC

Kings College in Cambridge

MICROSOFT'S GOALS WITH FARM

In the Microsoft "bing" search engine, they had a need to store various kinds of objects that represent common searches and results. These objects are small, but they have so many of them that in total, it represents a big-data use case.

The Microsoft research FaRM project (based in Cambridge England) was asked to help solve this problem.

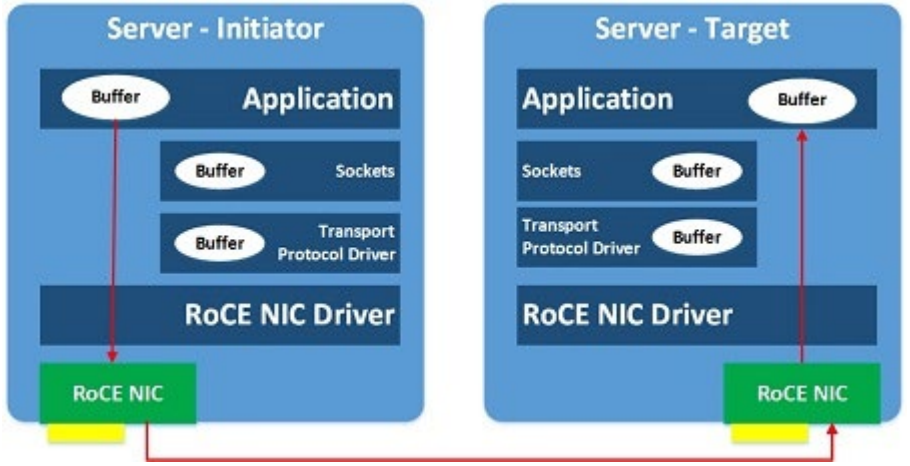
A FEW OBSERVATIONS THEY MADE

The felt they should try to leverage new kinds of hardware.

- The specific option that interested them was *remote direct memory access* networking, also called RDMA.
- RDMA makes the whole data center into a large NUMA system. All the memory on every machine can potentially be shared over the RDMA network and accessed from any other machine.

RDMA had never been used outside of supercomputers

RDMA HARDWARE

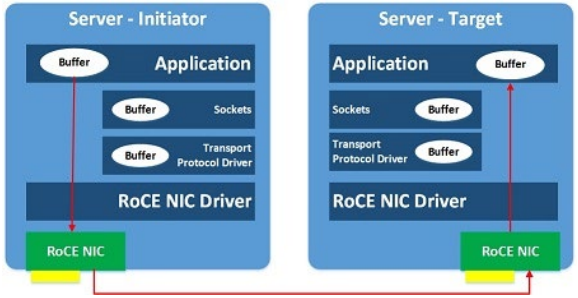


Emerged in the 1990s for high performance computers.

It has two aspects

- It moves a protocol like TCP *into the network interface hardware*. A can send to B without needing help from the kernel to provide end-to-end reliability: the network card (NIC) does all the work!
- There is a way to read or write memory directly: A can write into B's memory, or read from B's memory.

RDMA LIMITATIONS

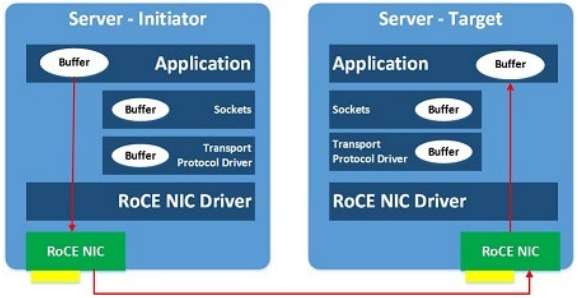


Early versions didn't run on a normal *optical* ethernet.

They used a similar kind of network, called Infiniband, but it has features that ethernet lacks.

- We saw that normal networks drop packets to tell TCP about overload
- **Infiniband never drops packets.** Instead, to send a packet a sender must get "credits" from the next machine in the route. These say "I have reserved space for *n* packets from you."
- Hop by hop, packets are moved reliably. In fact loss can occur, and if so RDMA will retransmit, but it is exceptionally rare.

RDMA ON RoCE

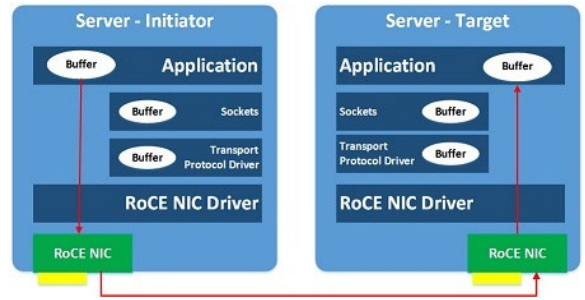


RDMA on Coverged Ethernet (RoCE) addresses this limitation.

It moves RDMA over to a normal TCP/IP optical network, but only for use within a single data center at a time. Infiniband is not needed – which means you don't need extra cables.

Microsoft was hoping to use RDMA in this "form" because they didn't want to rewire their Azure data centers.

HOW FAST IS RDMA?

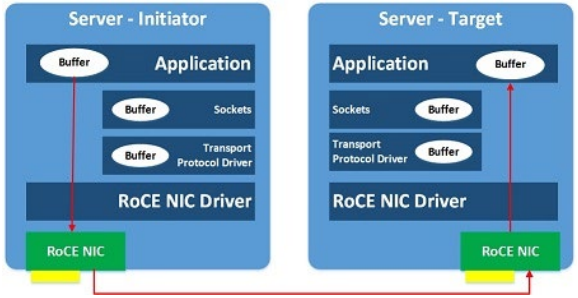


Similar to a NUMA memory on your own computer!

With the direct read or write option ("one sided RDMA"):

- It takes about 0.75us for A to write a byte into B's memory
- Bandwidth can be 100 to 200 Gbits/second (12.5 – 25 GBytes/s)
- This is 2 to 10x faster than *memcpy* on a NUMA machine!
- There is also less "overhead" in the form of acks and nacks. RDMA does need packets for the credits, but that's the only overhead.

THEIR IDEA



Purchase a new hardware unit from Mellanox that runs RDMA on RoCE cables. Microsoft didn't want to use Infiniband.

Create a pool of FaRM servers, which would hold the storage. Stands for "Fast Remote Memory"

The servers don't really do very much work. The clients do the real work of reading and writing.

… BUT COMPLICATIONS ENSUED

A "rocky" road!

The hardware didn't work very well, at first.

- Getting RDMA to work on normal ethernet was unexpectedly hard. RoCE is pronounced "rocky", perhaps for this reason.
- Solving the problem involved major hardware upgrades to the datacenter routers and switches, which now have to carry both RDMA and normal TCP/IP packets.

It cost millions of dollars, but now Microsoft has RDMA everywhere.

IT IS EASY TO LOSE THE BENEFIT OF RDMA

One idea was to build a protocol like GRPC over RDMA.

When Microsoft's FaRM people tried this, it added too much overhead. RDMA lost its advantage.

To leverage RDMA, we want server S to say to its client, A, "you may read and write directly in my memory". Then A can just put data into S's memory, or read it out.

THEY DECIDED TO IMPLEMENT A FAST SHARED MEMORY

The plan was to use the direct-memory access form of RDMA.

- **Insert(addr,object)** by A on server S would just reach into the memory of S and write the object there.
- **Fetch(addr)** would reach out and fetch the object.

A FaRM **address** is a pair: 32-bits to identify the server, and 32-bits giving the offset inside its memory (notice: 8 bytes).
The **object** is a byte array of some length.

DUE TO INTEREST FROM USERS, THEY ADDED A KEY-VALUE "DISTRIBUTED HASH TABLE"

The Bing developers preferred a memcached model. They like to think of "keys", not addresses. The solution is to use hashing: using std::hash, we can map a key to a pseudo-random number.

This works even if the key is std::string.

So we have a giant memory that holds *objects*.

… THEY ALSO HANDLE COMPLEX OBJECTS

One case that arises is when there some Bing object has many fields, or holds an array of smaller objects.

If the object name is "/bing/objects/1234", it can just be mapped to have keys like "/bing/objects/1234/field-1", … etc.

Note that the data might scatter over many servers. But in a way this is good: a chance for parallelism on reads and writes!

A PROBLEM ARISES!

What if two processes on different machines access the same data? One might be updating it when the other is reading it.

Or they might both try to update the object at the same time.

These issues are rare, but we can't risk buggy behavior. FaRM needs a form of critical section.

LOCKING OR MONITORS?

In a C++ process with multiple threads, we use mutex locks and monitors for cases like these.

But in FaRM the processes are on different machines.

Distributed locking is just too expensive.

DOWN TO BASICS: WE NEED ATOMICITY

We learned about C++ atomics. Atomicity can be defined for a set of updates, too:

- We need them to occur in an all or nothing manner. [**ordering**]
- If two threads try to update the same thing, one should run before the other (and finish) before the other can run. [**isolation**]
- Data shouldn't be lost if something crashes. [**durability**]

CONCEPT: A TRANSACTIONAL WRITE

We say that an operation is an *atomic transaction* if it combines a series of reads and updates into a single indivisible action.

The transaction has multiple steps (the individual reads and writes, or get and puts). Software creates an illusion that they occur all at once.

Readers always see the system as if no updates were underway. Updates seem to occur one by one.

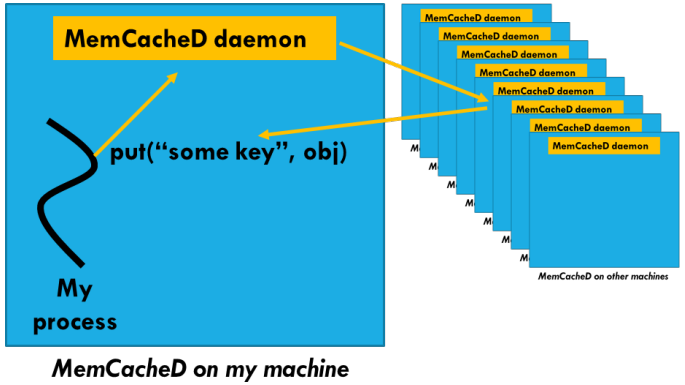
FARM TRANSACTIONS

They decided to support two kinds of transactions

1. An atomic update that will replace a series of key-value pairs with a new set of key-value pairs.

2. An atomic read that will read a series of key-value pairs all as a single atomic operation.

REMINDER OF THE PICTURE



Recall that a key-value store is a two-level hashing scheme:

1. Find the proper server for a given key.

2. Then within that server, do an O(1) lookup in a hashed structure, like the C++ std::unordered_set

FARM VARIANT

Same idea, but now a single Bing update could require many concurrent put operations, or many concurrent get operations.

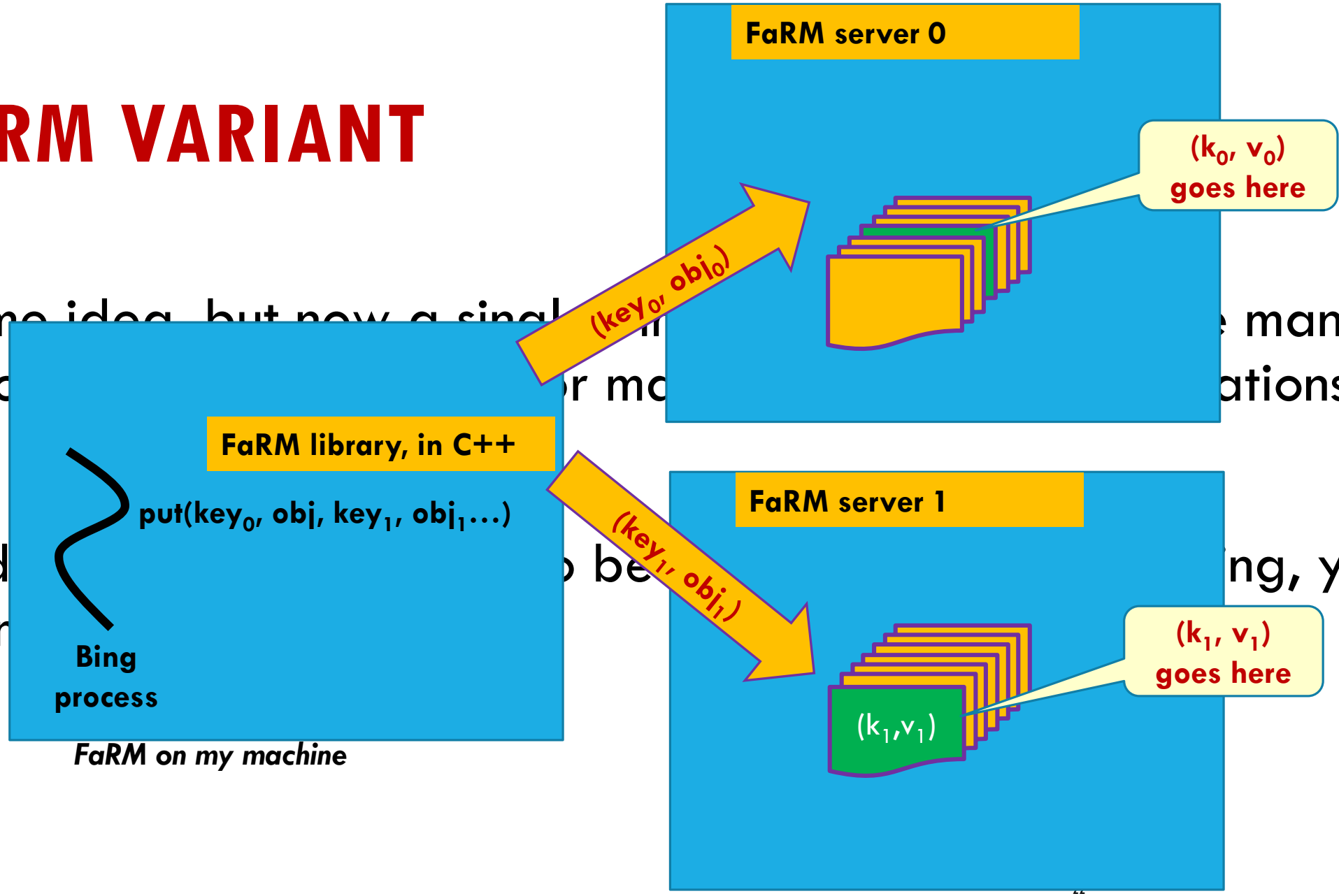
Also, RDMA is direct, so we won't have any daemon

And we want updates to be concurrent, lock-free, yet atomic.

FARM VARIANT

Same idea, but now a sing… many
conc… r ma… ations.

And … be… ng, yet
ator…



HOW THEY SOLVED THIS

Microsoft came up with a way to write large objects without needing locks.

They also found a hashed data structure that has very limited needs for locking.

FIRST, THE TRANSACTION MODEL

Let's first deal with the "many updates done as an atomic transaction" aspect.

Then we will worry about how multiple machines can safely write into a server concurrently without messing things up.

THE TRANSACTIONAL WRITE IDEA

Suppose that some object requires *k* updates. FaRM breaks the data into chunks of size 60 bytes, adds a 4-byte "version id" to each update, in a hidden field (a "header"), obtaining 64-byte records.

version update_0 ***version*** update_1 …. ***version*** update_k

64 bytes 64 bytes…

In FaRM the version number is a hash of who did the write and the transaction id. The end user will never see this number.

THE TRANSACTION WRITE IDEA, CONTINUED

The basic algorithm FaRM uses is this:

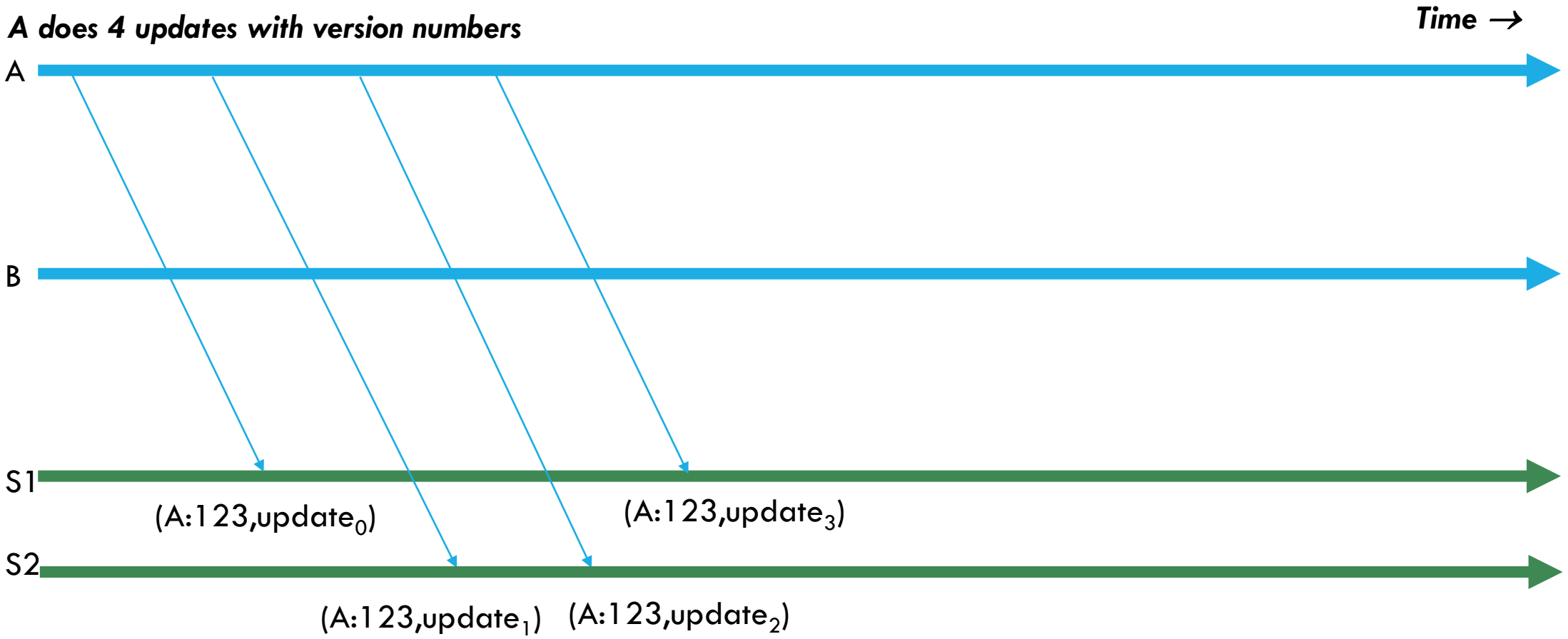
To write X, A first tags each update with a version number

Now it writes all its updates.

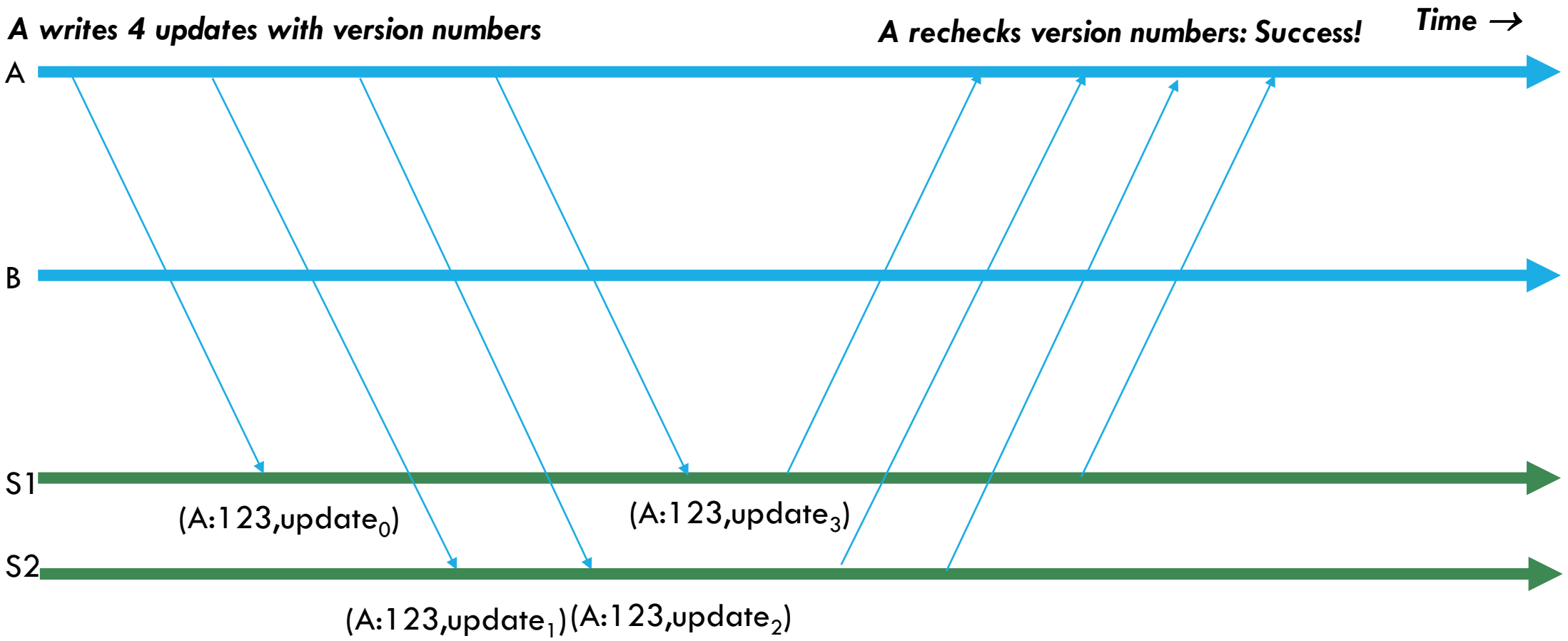
Then A goes back to check that the version numbers at the end are the same as the version numbers it wrote

Same if some process reads the data

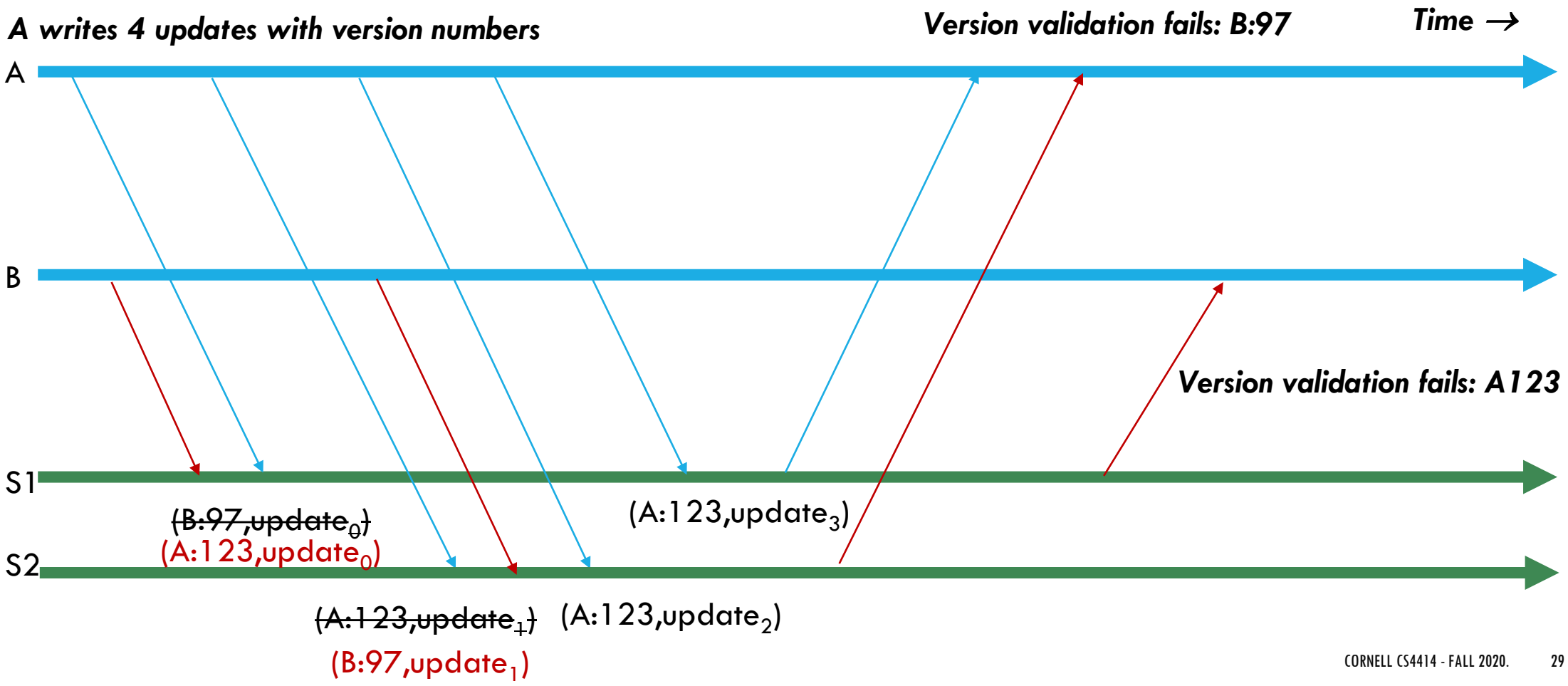
A TRANSACTIONAL WRITE



A TRANSACTIONAL WRITE: NORMAL CASE



A TRANSACTIONAL WRITE: CONFLICT CASE



BASIC RULE

If the version numbers all match, the object is intact.

If the version numbers don't match, something disrupted A's read or write, and it must retry the same request.

WHY COULD THEY FAIL TO MATCH?

If A and B write at the same time, they use different version numbers. There are three possible cases:

1. A writes first, then B. Version numbers match but are the ones B wrote.
2. B writes first, then A. Same, but now we see A's versions.
3. They overlapped, leaving some unmatched version numbers.

IF A OR B IS "SURPRISED" BY THE RESULT

Pick a random small number, delay by this amount of time.

They will pause by different amounts: A "random backoff". Perhaps, A doesn't pause at all, but B waits 2us

FaRM should rarely see conflict. When a conflict does occur, A retries instantly. B pauses, then later will wake up and do its write.

WHY ISN'T LOCKING NEEDED?

FaRM does not require any locks for reads or writes provided that conflicts occur rarely.

This is because the validation step will usually succeed, and the backoff step will rarely even need to run.

But there are a few objects where conflicts are more common.

LOCKING

For heavily contended-for objects, or for cases where an overwrite should not be allowed, they implemented per-key locking using a new RDMA test-and-set feature.

These “lock objects” allow Bing to get true mutual exclusion if genuinely needed, but Bing developers were urged to use them very rarely – they harm performance otherwise.

HOW A UPDATES SOME RECORD IN S

A prepares the 64-byte object.

Now A figures out which server to talk to: a first hashing step.

And then A figures out which array element to access: a *second* hash and modulus operation, modulo the size of the table in S. A uses RDMA to read or write directly into this memory region.

WHAT IF A AND B "COLLIDE" AT THIS STEP?

For a single 64-byte record, RDMA itself handles atomicity.

Either A will "win" and go first, and B will run second, or vice versa. The hardware does it and FaRM has no need for any kind of special logic.

If the update is part of a multi-write transaction, the transaction logic we saw on slides 32-25 will resolve any conflicts.

HASH COLLISIONS

Another puzzle is this.

Suppose that A and B are accessing S *using different keys*. The objects are different. No conflict has arisen here.

Yet those keys could hash to the identical slot in memory. This is called a *hash collision*. We don't want the objects to overwrite one-another: we need a way to keep both!

HOPSCOTCH HASHING

Hopscotch hashing is done once FaRM already knows which storage server will hold a particular (key,update) pair.

FaRM hashes to find the slot an item should go into, but first reads the slot to see if it is full. It asks: are we updating a value or inserting some other key?

Replacement can occur in place. To insert a different key,FaRM scans the region "around" the slot. The new item goes into the closest empty slot.

HOW TO DO A READ? HOPSCOTCH READS!

We need to find the slot containing an object with the key.

We start by hashing to the desired slot. Often the item is there. FaRM can tell by just checking the key for a match.

If not, FaRM reads the region around the slot, checking for the key (recall that the updates are stored with their keys). If this pattern encounters an empty slot, the item isn't in the store.

OUTCOME?

Now A and B can treat the pool of storage servers as a NUMA memory unit. In fact keys are like memory addresses.

S helps with RDMA setup, but then A and B can read/write concurrently without help from S.

The algorithm is lock-free except for high-contention objects.

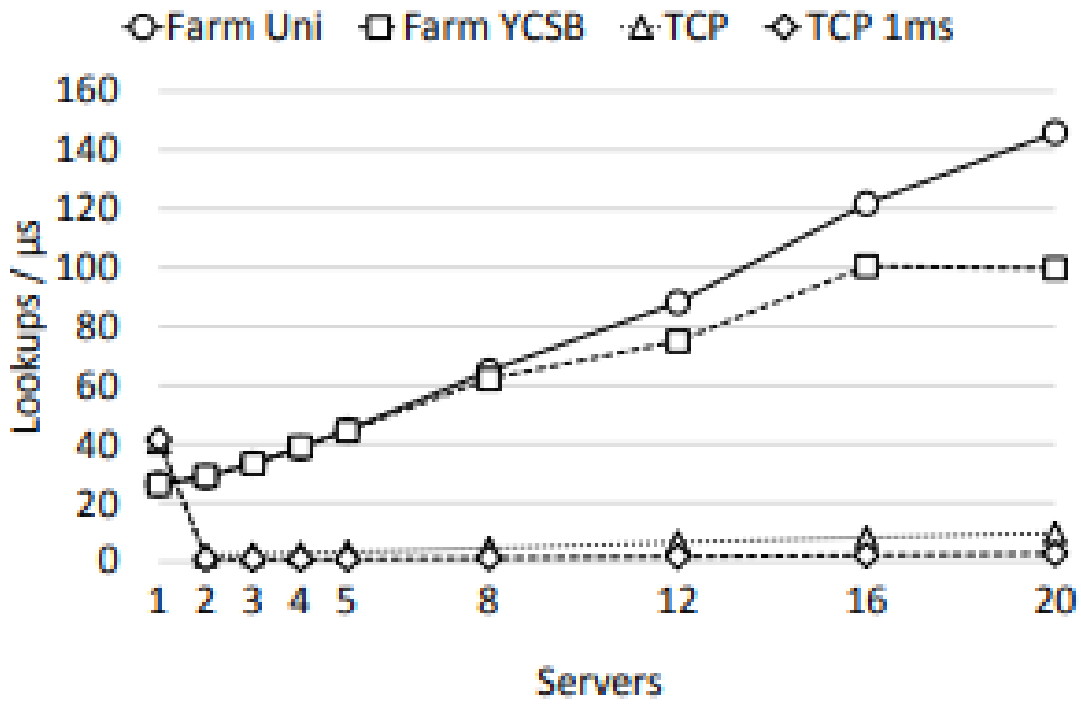
FARM PERFORMANCE: LOOKUP RATE

FaRM Uni: FaRM with 120M (k,v) pairs, accessed uniformly at random.

FaRM YCSB: Items accessed according to the YCSB benchmark popularity pattern.

TCP: FaRM running over a hand-tuned TCP-based protocol, similar to GRPC

TCP 1ms: FaRM on TCP with a 1ms response time guarantee



https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-dragojevic.pdf

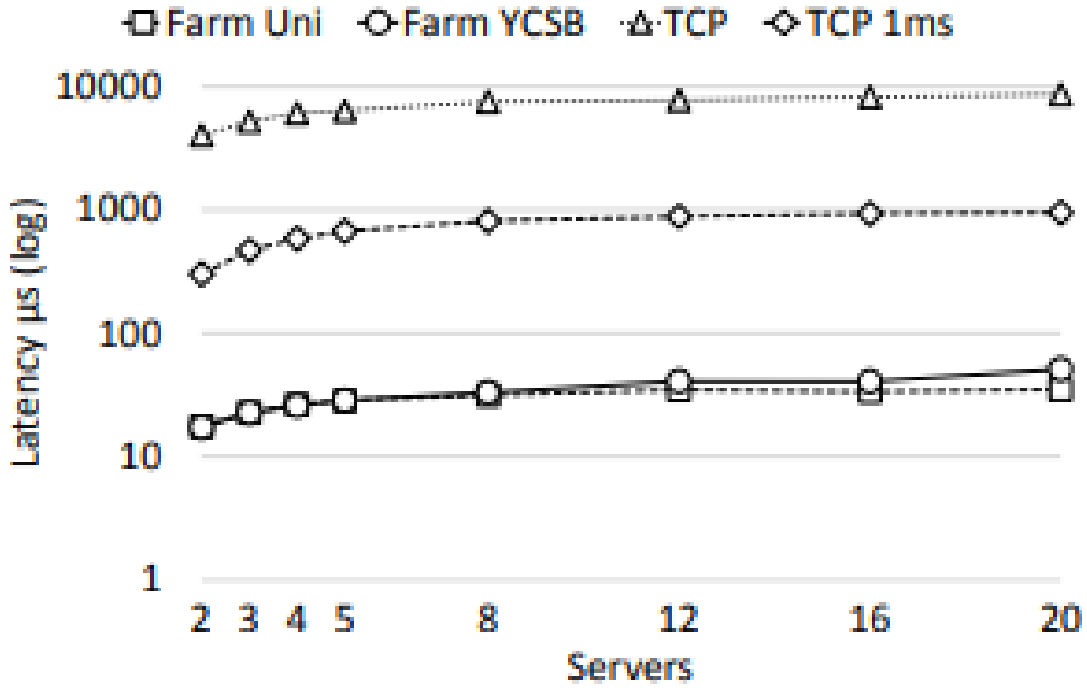
FARM PERFORMANCE: DELAY TO DO LOOKUP

FaRM Uni: FaRM with 120M (k,v) pairs, accessed uniformly at random.

FaRM YCSB: Items accessed according to the YCSB benchmark popularity pattern.

TCP: FaRM running over a hand-tuned TCP-based protocol, similar to GRPC

TCP 1ms: FaRM on TCP with a 1ms response time guarantee



FARM PERFORMANCE: TRANSACTIONAL READ

FaRM Uni: FaRM with 120M (k,v) pairs, accessed uniformly at random.

FaRM YCSB: Items accessed according to the YCSB benchmark popularity pattern.

TCP: FaRM running over a hand-tuned TCP-based protocol, similar to GRPC

TCP 1ms: FaRM on TCP with a 1ms response time guarantee

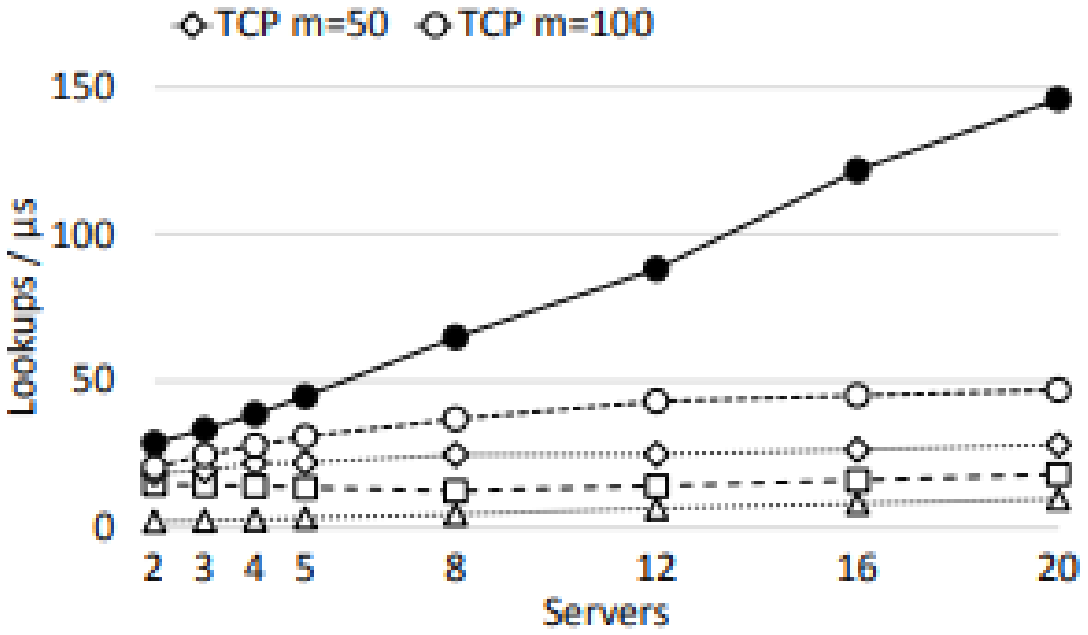


Figure 13: Key-value store: multi-get scalability

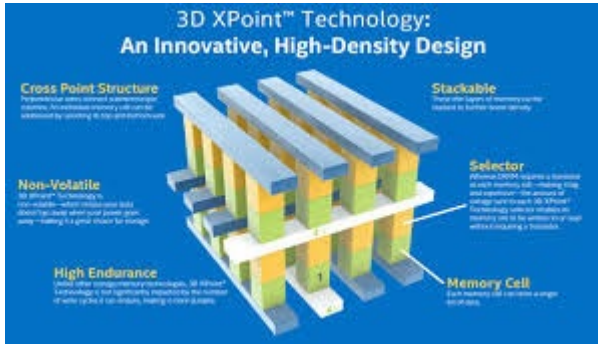
BACKUP

Bing also needed some form of backup.

Objects can be lost, rarely, but in general Bing wants the data it stored in FaRM to be there after a server crashes, then restarts.

For this, Microsoft had a few ideas

3-D XPOINT MEMORY



Normal memory will be lost in the event of a crash. Replication only helps if the backup doesn't crash, too. Some Bing uses need more.

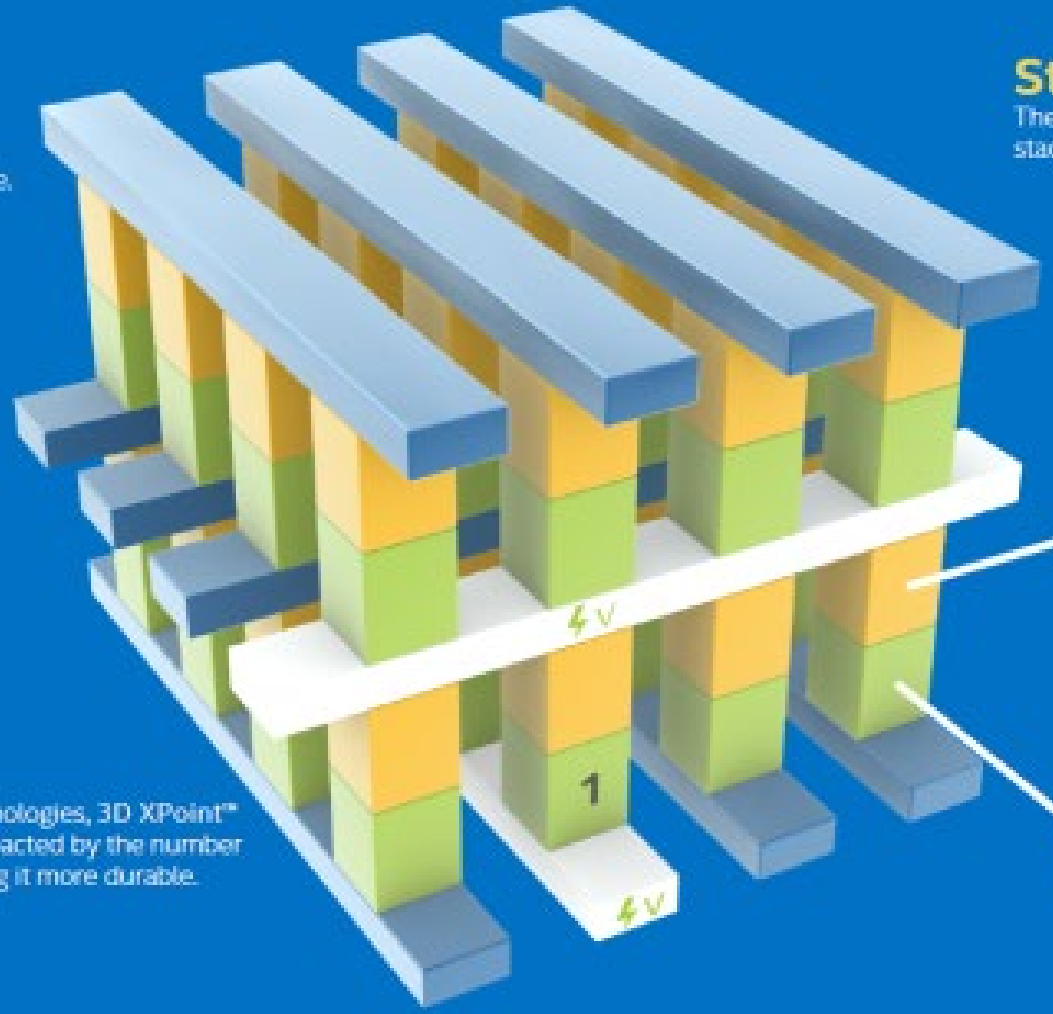
The FaRM project decided to experiment with a new form of memory called 3-D XPoint, based on phase-change memory hardware (a novel layered semiconductor technology)

The idea was that anything in the FaRM memory would survive crashes and just be "in memory" on restart!

3D XPoint™ Technology:
An Innovative, High-Density Design

Cross Point Structure
Perpendicular wires connect submicroscopic columns. An individual memory cell can be addressed by selecting its top and bottom wire.

Stackable
These thin layers of memory can be stacked to further boost density.



Non-Volatile
3D XPoint™ Technology is non-volatile—which means your data doesn't go away when your power goes away—making it a great choice for storage.

Selector
Whereas DRAM requires a transistor at each memory cell—making it big and expensive—the amount of voltage sent to each 3D XPoint™ Technology selector enables its memory cell to be written to or read without requiring a transistor.

High Endurance
Unlike other storage memory technologies, 3D XPoint™ Technology is not significantly impacted by the number of write cycles it can endure, making it more durable.

Memory Cell
Each memory cell can store a single bit of data.

… NO LUCK

3-D XPoint took years longer than expected to reach the market.

Sample units were much slower than expected when used as memory, although they were impressive as disks. (Slow DMA transfer rates)

Even today, this technology can't just be used like normal memory!

FALLBACK: WRITE-BEFORE LOG

FaRM was forced to use a fallback: storage drives based on a fast form of flash memory (similar to a USB)

The FaRM servers watch for updated portions of the key-value store and log them to the server in a continuous stream.

FALLBACK: WRITE-BEFORE LOGGING

For sensitive data that cannot be lost, FaRM has a way to pause the writer to wait until the log is written to disk.

Called “write-before logging”: First log the update, then finalize it and allow readers to see the data.

This way, a writer can be sure the data won’t be lost.

IT WAS TOO SLOW!

250TB for only $500,000…

NAND storage speeds are surprisingly variable: some operations are fast, some slow.

Microsoft realized that write-ahead logging would emerge as a bottleneck if they didn't find a solution to this.

THEY ADDED TWO OPTIONS

Some Bing uses only need high availability, not persistence. For these, they replicated FaRM.

Data is still held purely in memory, but now there is a copy in an active replica server: two copies of each item. If that machine doesn't depend on the same hardware in any way, it shouldn't crash even if the primary copy goes down.

BATTERY-BACKED CACHE FOR FLASH DRIVES

A second option was to use a storage unit that has a battery-backed RAM cache in front of the flash-memory storage.

A DMA write first goes into the battery-backed memory: a very fast transfer. Now the data is "safe".

The write to flash memory occurs soon after, but no need to pause unless the RAM memory cache fills up. Even if power is lost, the battery-backup will allow the device to finish the writes.

BING REALLY USES FARM

Not every "academic research" project has impact on big corporate products like Bing. FaRM is unusually impactful.

But normal Azure users can't access FaRM (yet) and RDMA is used only by special services like FaRM – not you and me!

NOTICE THE SIMILARITY TO C++ IDEAS IN A SINGLE COMPUTER!

Many of these same concepts are relevant to C++ with threads.

In fact, this is deliberate. The FaRM team members are C++ developers and wanted FaRM to feel natural.

People always prefer the same ideas in new settings… not new ideas, unless there is no choice!

SUMMARY

Modern applications often run into big-data uses

Microsoft created FaRM as an RDMA-enabled shared memory. Later it evolved for Bing: the developers preferred a key-value model.

The extreme efficiency of FaRM comes from the mapping to RDMA hardware. But deploying this cutting-edge hardware was hard.