Ken Birman

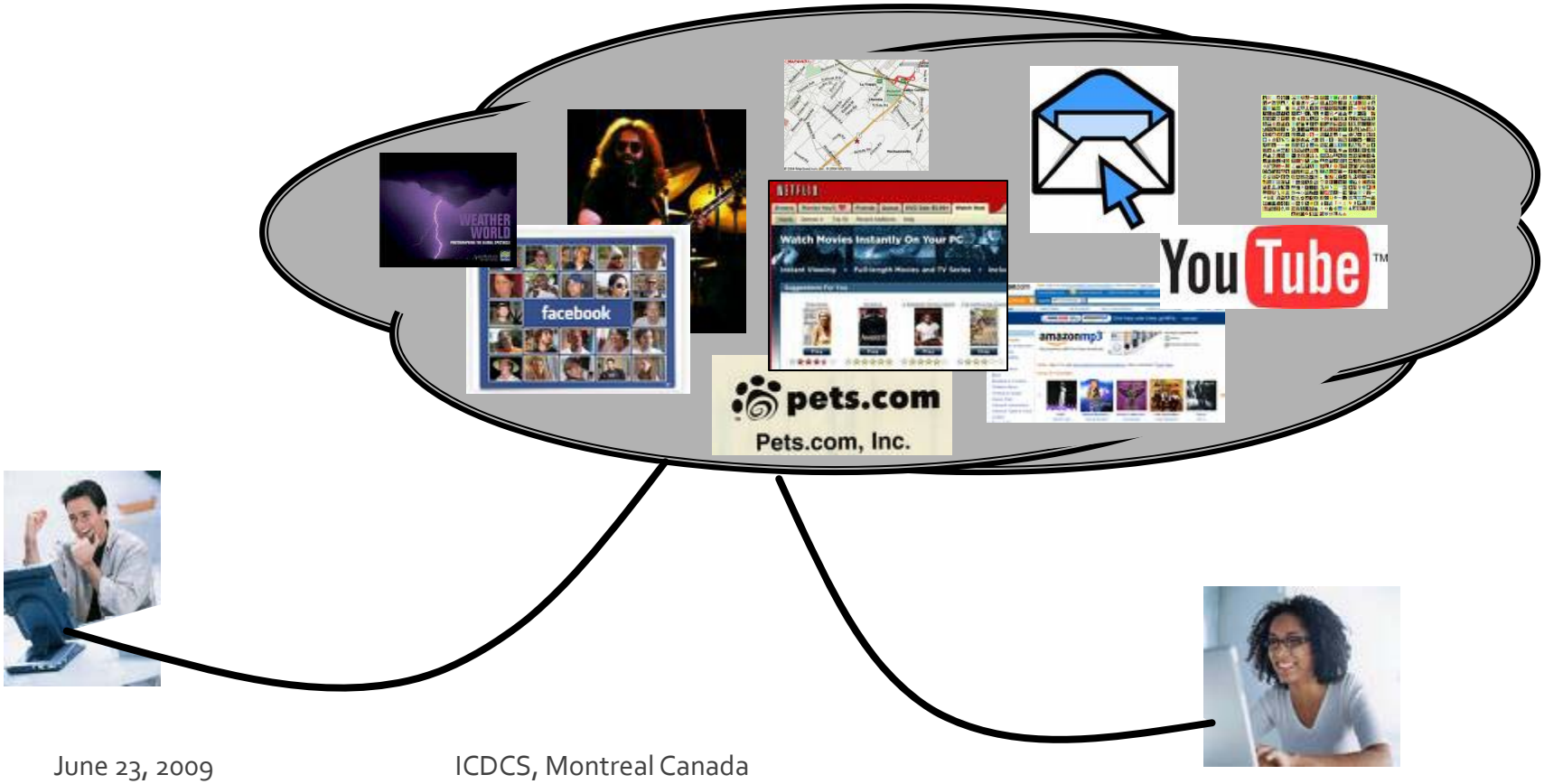
Rethinking Multicast for Massive Platforms and Applications

The Old World and the New

- My career has been focused on multicast
  - Mostly, in support of service replication
- … and this was important in the "old world"
  - Client-server computing, dominated by databases
  - ACID consistency guarantees were the key

    The "virtual synchrony" model is a form of ACID property
- But a new world has displaced the old one

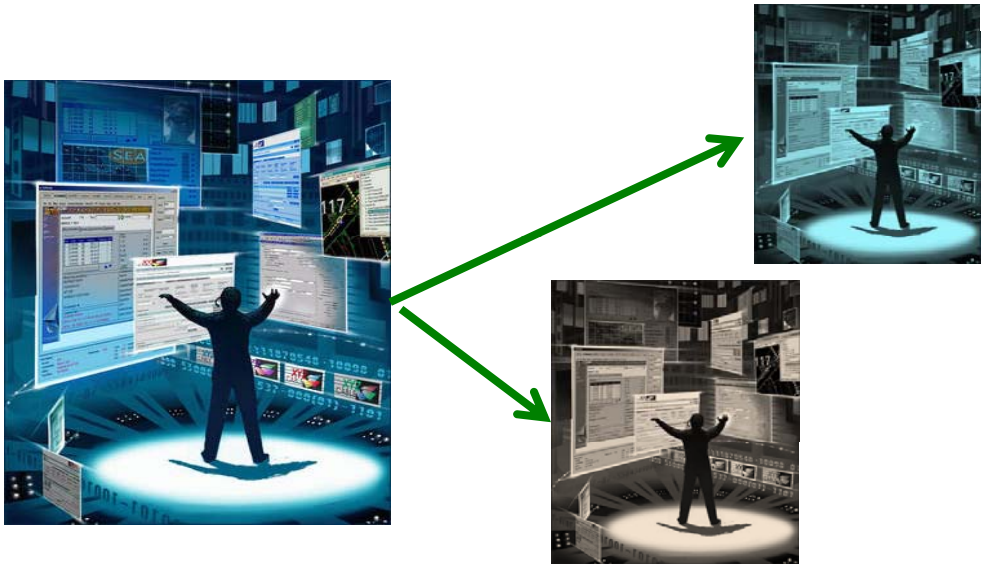
Massive Cloud Platforms



… and a Live, Collaborative Web

- ***Edge Mashups***

Simple ways to create and share collaboration and social network applications



[http://liveobjects.cs.cornell.edu]

- Examples: Google "Wave", Javascript/AJAX, Silverlight, Java Fx, Adobe FLEX and AIR, Live Objects, etc….

The Old World and the New

- **Old world:** we replicated servers for speed and availability, but maintained consistency
- **New world:** *scalability* matters most of all
  - *Lots of replication, but weak consistency*

Cloud Computing Puzzle

- ***Why can't we have both?***
  - Scalability
  - Consistency too

RoadMap

We'll ask…

- Why do reliability/consistency mechanisms scale poorly?
  - Focus on replication supported by multicast
- Can we do something about it?
  - For example, can we fix the "whole mess"?
- How would a truly scalable platform look?

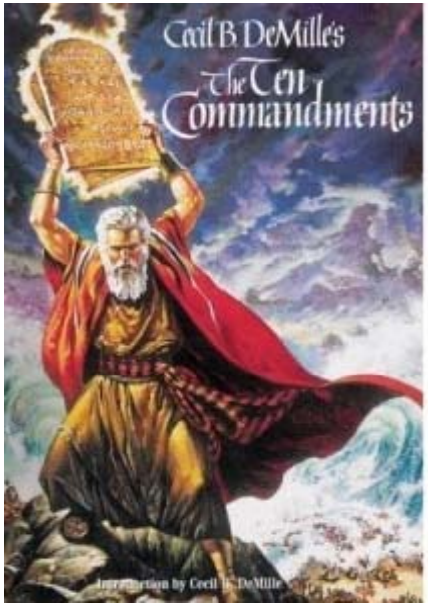
The Wisdom of the Sages

eBay's Five Commandments

- As described by Randy Shoup at LADIS 2008

Thou shalt…

1. **Partition Everything**
2. **Use Asynchrony Everywhere**
3. **Automate Everything**
4. **Remember: Everything Fails**
5. **Embrace Inconsistency**



Vogels at the Helm

- Werner Vogels is CTO at Amazon.com…
- His first act? ***He banned multicast!***
  - Amazon was troubled by platform instability
  - Vogels decreed: all communication via SOAP/TCP

- This was slower… but
  - ***Stability mattered more than speed***

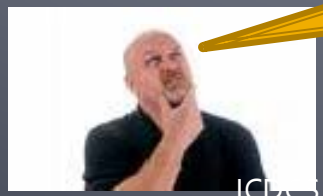
James Hamilton's advice

- Key to scalability is decoupling, loosest possible synchronization
- *Any* synchronized mechanism is a risk
  - His approach: create a committee
  - Anyone who wants to deploy a highly consistent mechanism needs committee approval

…. They don't meet very often

Consistency

Consistency? I hate it!

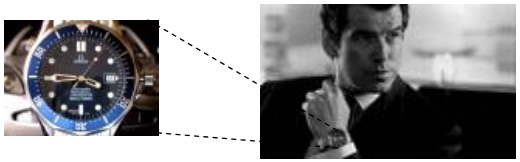


They all hate consistency!

- This is the common thread
- All three guys
  - Really build massive data centers, that work
  - **And are opposed to "consistency mechanisms"**

What's consistency?

A <u>consistent</u> distributed system will often have many components, but users observe behavior indistinguishable from that of a single-component reference system



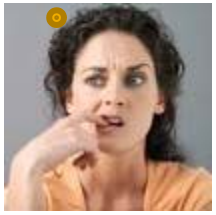
Reference Model

Implementation

Dangers of Inconsistency

My account is overdrawn?
That can't be right!

- Inconsistency causes bugs
  - Clients would never be able to trust servers… a free-for-all



- Weak or "best effort" consistency?
  - Strong security guarantees demand consistency
  - Would you trust a medical electronic-health records system or a bank that used "weak consistency" for better scalability?

Why do CTOs hate consistency?

- They think consistency-mechanisms are capable of destabilizing the whole data center
- They also doubt that these mechanisms can be made to scale adequately

An agenda with two parts

1. First, we'll "fix the network"
   - Some network mechanisms commonly used by consistency protocols are prone to meltdowns
   - These disrupt everyone, *not just the application that actually triggered the problem*

2. Next, we'll ask whether consistency properties can be made to scale better

Networks that actually work

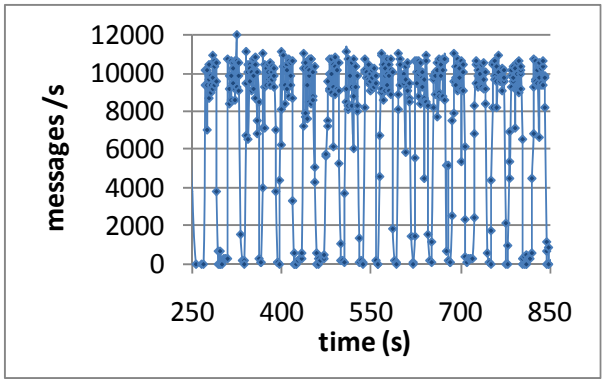
Fixing the communication layer

Definition: Network

- For this part of the talk
  - TCP
  - UDP
  - IP multicast (IPMC)
- Assertion: Data center networks "broke" around 2001 (rollout of 1-10 Gbit networks)

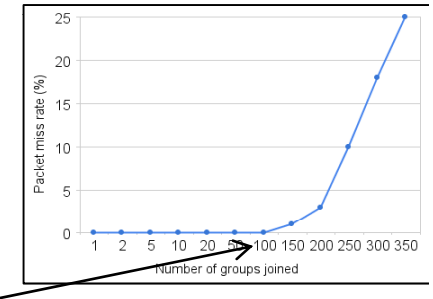
Anatomy of a meltdown

- A "blend" of stories (eBay, Amazon, Yahoo):
  - Pub-sub message bus very popular. System scaled up. Rolled out a faster ethernet.
  - Product uses IPMC to accelerate sending
  - All goes well until one day, under heavy load, loss rates spike, triggering collapse
- Oscillation observed



Use of IPMC: Risk/Benefit tradeoff

- Without IPMC?
  - Pub/sub strugglesto disseminate data rapidly
  - This inevitably will limit scalability



- But IPMC itself scales poorly!
  - Routers and NICs become promiscuous if we use large numbers of multicast addresses [Tock]
  - Then *everyone gets every IPMC packet*

Why did loss rates spike?

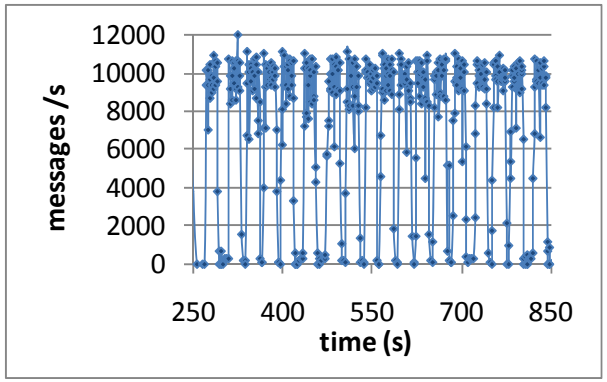
- "Normal" communication is 1:1
  - The network is much faster than any NIC
  - But TCP windowing keeps rates pretty balanced
- If IPMC becomes promiscuous, receivers get lots of junk
  - It shows up ***fast***, mixed with valid data
  - This swamps the receivers, which drop packets
  - … including good packets

Why does loss trigger oscillation?

1. System gets larger and more loaded
2. Generalized loss starts to occur
3. Causes a massive spike of NAK messages. Retransmissions just make things worse

   meltdown!

4. After 90 seconds, the pub-sub product resets



Are there other causes?

1. Disruptive events are more common as size increases, causing system to slow down
2. Overheads often grow in system size… *sending system off a cliff*
3. Oscillatory behavior surprisingly common
   - Perhaps spontaneous
   - …or triggered by some form of churn
   - …or a convoy/bottleneck
   - …or a priority inversion

Connection to Consistency?

- A consistent system fights to maintain some guarantee: a feedback loop
  - Thus when the network collapses, the consistency property is often seen to be at fault
  - Had we not cared about consistency, feedback instability would not have occurred

So… consistency is dangerous!

Fixing the Network

Legislate against IPMC and UDP?

- This is more common than one might expect
- Our premise?
  - Scalable consistency protocols will need to leverage IPMC and UDP
  - Today they can't because both are poor citizens

Fixing IPMC? Aggregation!

- Recall: IPMC became promiscuous because *too many multicast channels were used*
- My not *aggregate* (combine) IPMC channels?
  - When two channels have similar receiver sets, combine them into one channel
  - Filter (discard) unwanted extra messages

Channel Aggregation

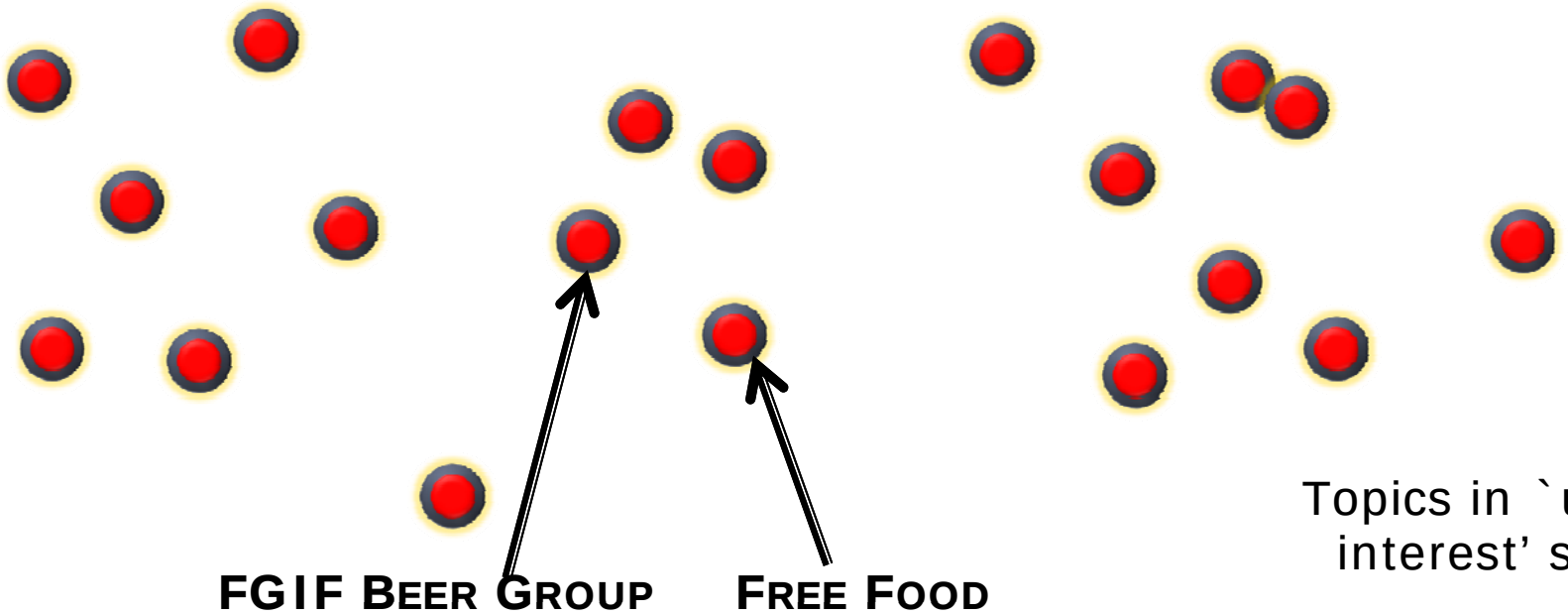
- Algorithm by Vigfusson, Tock [HotNets 09]; uses a k-means clustering algorithm
  - Generalized problem is NP complete
  - But heuristic works well in practice

Optimization Questions

- Assign IPMC and unicast addresses s.t.
  - $\leq \alpha$ % receiver filtering (hard)
  - (1) Min. network traffic
  - $\leq M$ # IPMC addresses (hard)

- Prefers sender load over receiver load
- Intuitive control knobs as part of the policy

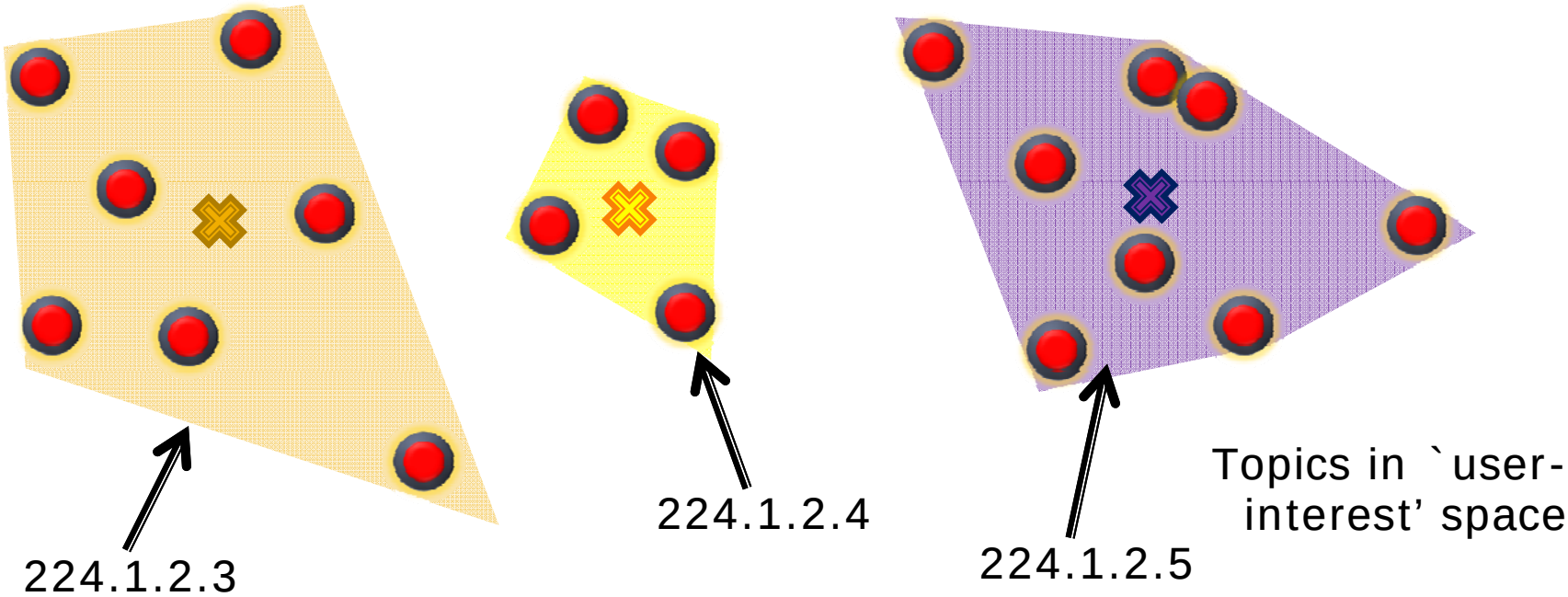
MCMD Heuristic



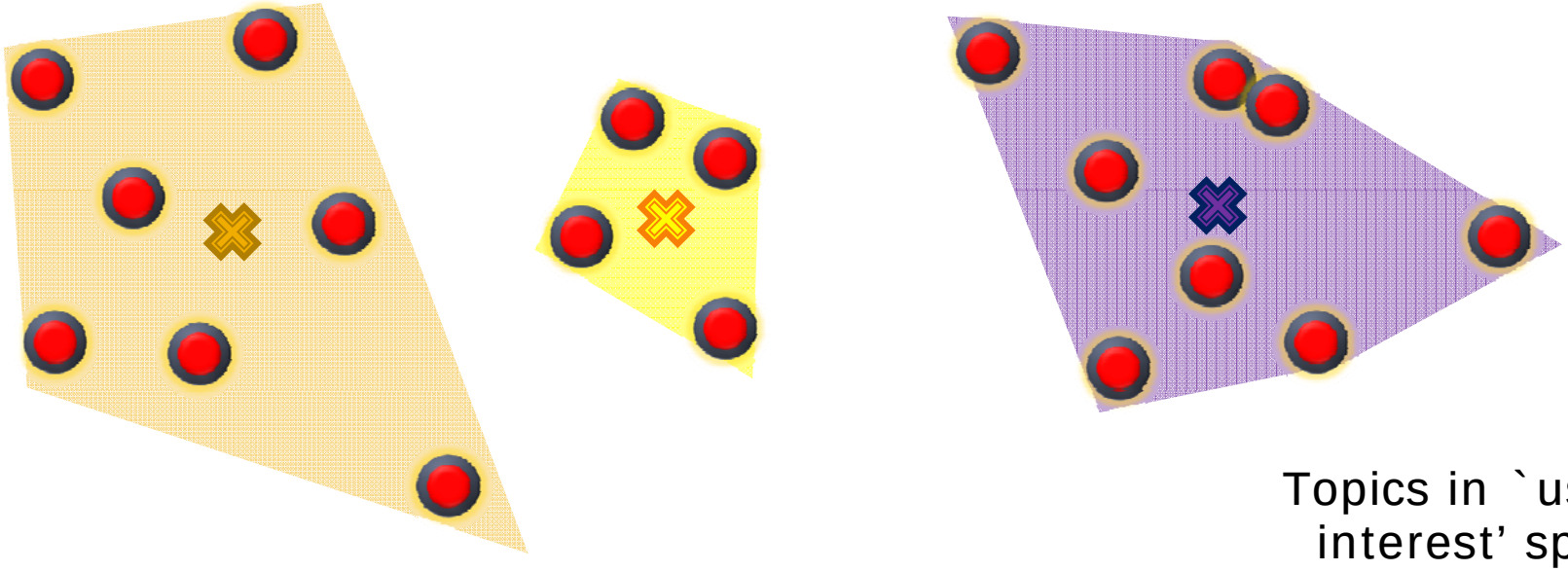
Topics in `user-interest' space

(0,1,1,1,1,0,1,0,0,0,1,1)

MCMD Heuristic



MCMD Heuristic

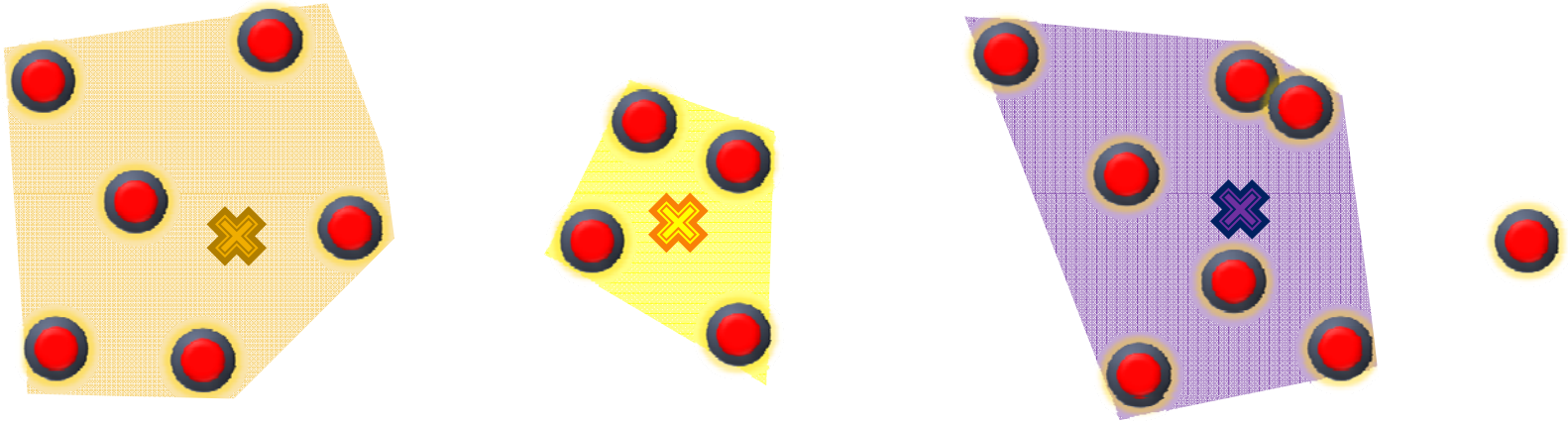


Topics in `user-interest' space

Sending cost:

Filtering cost:

MCMD Heuristic



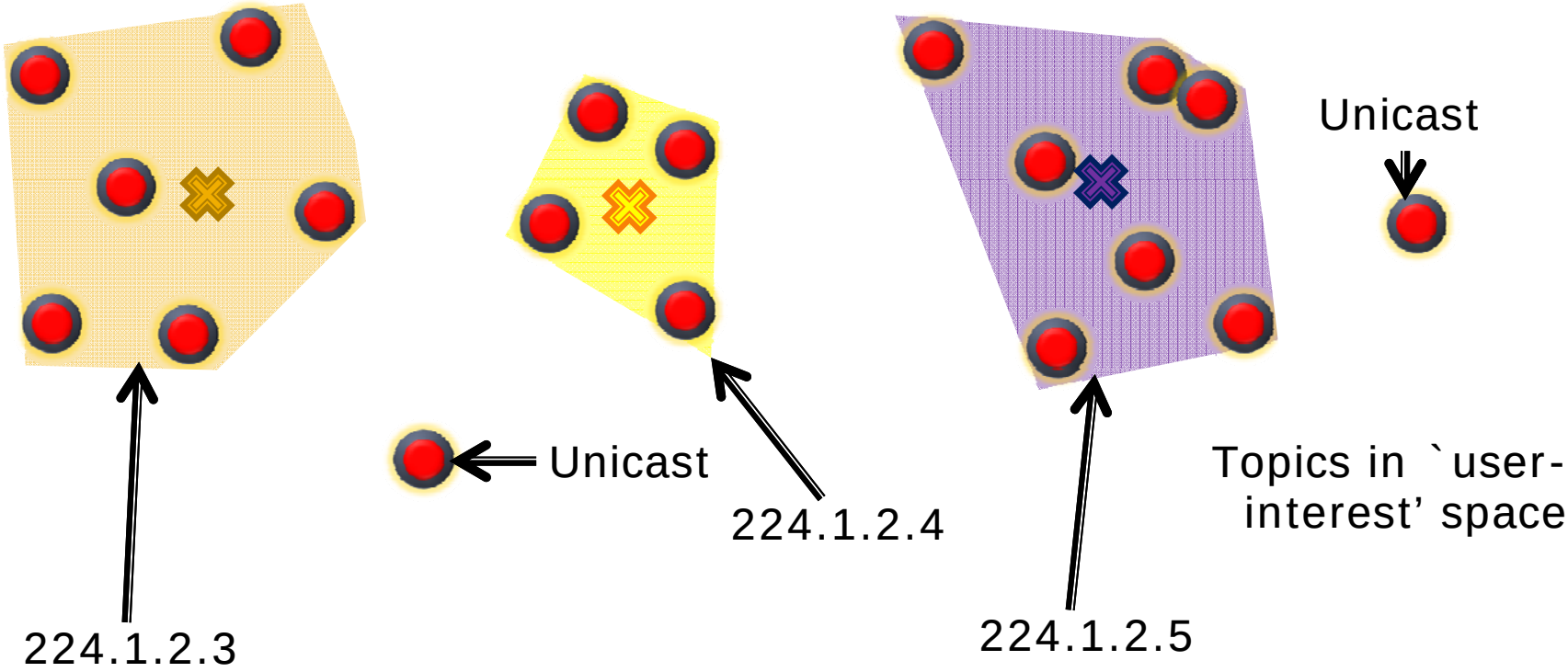
Unicast

Topics in `user-interest' space

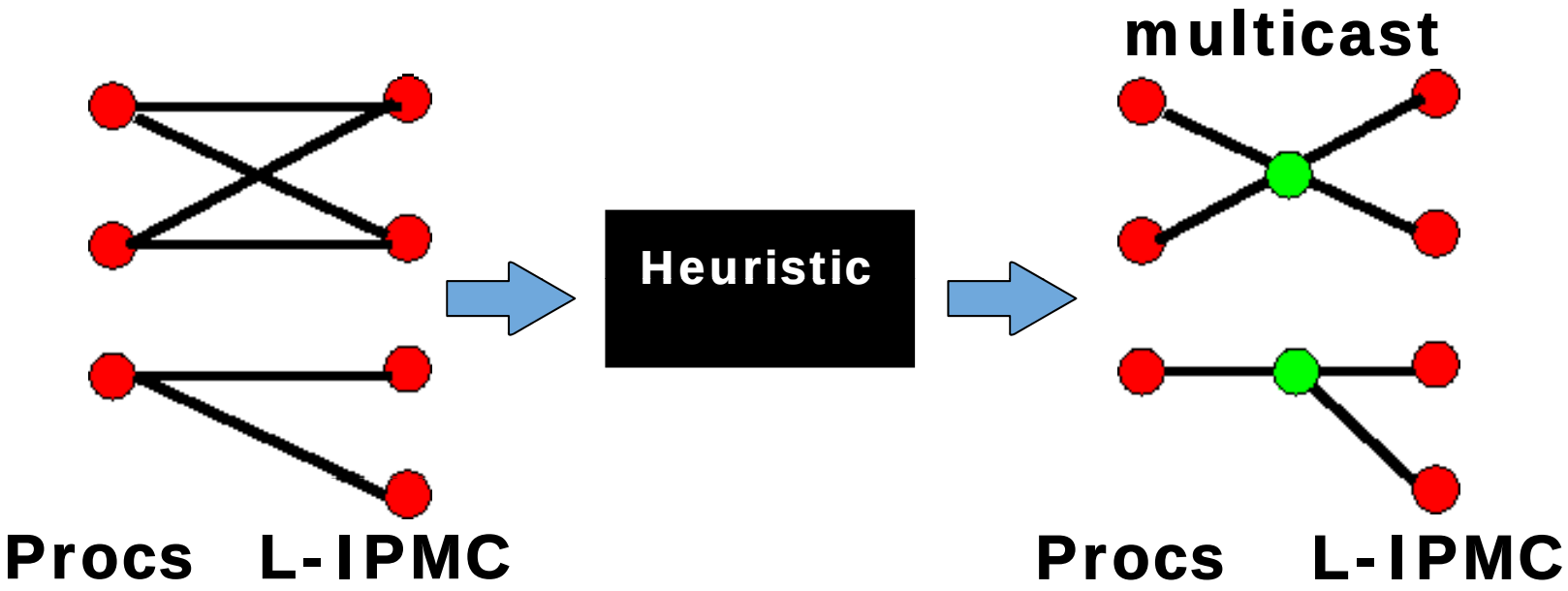
Sending cost:

Filtering cost:

MCMD Heuristic



Using the Solution



- Processes use “logical” IPMC addresses
- Dr. Multicast transparently maps these to true IPMC addresses or 1:1 UDP sends

Dr. Multicast: Only part of the story

- With Dr. Multicast, we can avoid overloading the hardware IPMC "resource"
- Hence won't see network switch to promiscuous behavior
- But very heavy multicast traffic could still provoke high rates of packet loss

Rate Control: AJIL

- Needed to prevent bursts of multicast from overrunning receivers
- AJIL protocol imposes limits on IPMC *rate*
  - AJIL monitors aggregated multicast rate
  - Uses an optimization rule to apportion bandwidth
  - If limit exceeded, user perceives a "slower" multicast channel

Before / After

- Without Dr. Multicast+AJIL
  - It was easily to accidentally exceed the hardware limits on IPMC use, triggering promiscuous mode
  - Thus: IPMC sometimes fell off a reliability cliff
- With Dr. Multicast+AJIL
  - We invisibly but inexpensively prevent such issues
  - IPMC becomes a reliable and friendly option

Next: Fix UDP too

- Based on Gossip Objects [Vigfusson LADIS '09]
  - Basic idea: *Gossip* protocols respect fixed maximum rate and packet size limits
  - Why not run UDP over gossip?
    - *UDP packet treated as a "rumor"; optimization policy decides precisely who to talk to, and when.*
- Yields a new kind of network "isolation"
  - TCP friendly… IPMC friendly… UDP friendly too
  - In effect: A new kind of VPN

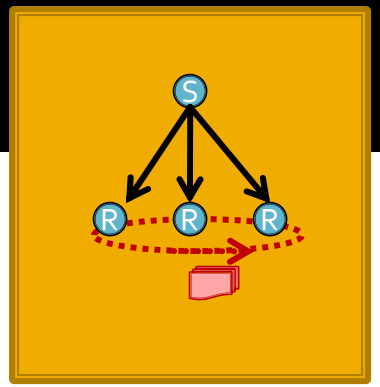
Hypothesis: A new kind of VPN

- If we can fix the network, so that it scales without collapsing, then we can fix our consistency protocols too
- Underlying the hypothesis: provable scalability is feasible *for some protocols*
  - We're not claiming that any random protocol can be magically made to scale properly

… enabling scalable multicast

- Designing a multicast protocol that scales
  - Hierarchical internal architecture
  - Minimal event handling costs, memory footprint
  - Avoid priority inversions
  - Can use IPMC now – it won't fall off cliffs
- Leaving us with a new issue: complexity
  - Reliability protocols are often complicated, and the things that let them scale sound complex too!

Example: Quicksilver Scalable Multicast [Ostrowski NCA 08]



- QSM works as just described
  - Sends each multicast using IPMC
  - Membership svc organizes nodes into rings (if many, a tree of rings), token circulates in rings
  - Token queries: "*what messages do you have*?"
    - A sees that B lacks **m**? A forwards m (via UDP).
    - Entire ring missing **m**? Recover from another ring
    - Everyone has the message? Garbage collect it
- If IPMC is highly reliable, QSM performs and scales incredibly well!

QSM Scalability

- Tested at up to 100Mbits/second (limit for our cheap cluster), 200 nodes, 8000 groups
  - Can saturate the network with useful traffic
  - Performance shows almost no degradation as number of members or groups is increased
  - Immune to oscillations even when "attacked"
  - Robust handling of membership churn
- *But QSM has weak "consistency properties"…*

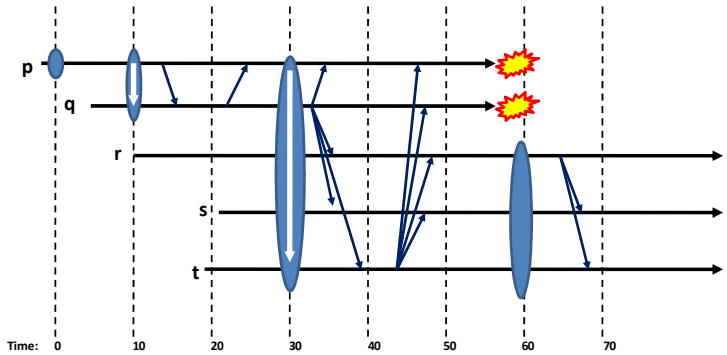
Properties Framework

Protocol properties

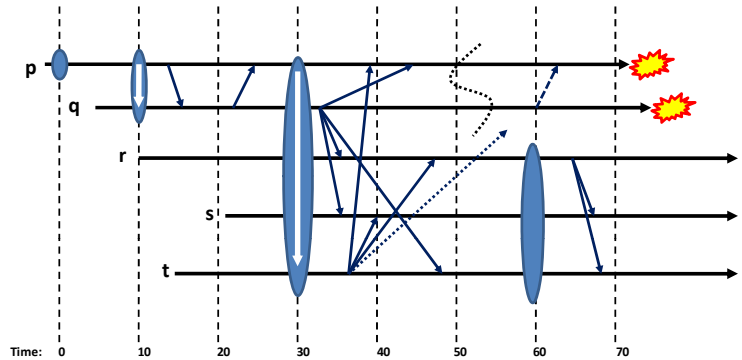
- A tool to reduce complexity
  - Write our protocols in a high-level language
  - Compiler will generate implementations that use a standardized hierarchical structure for scale
  - We call this structure the "Properties Framework"

- The framework is actually built using QSM: Quicksilver Scalable Multicast [Ostrowski: NCA 08]

Example: Virtual Synchrony



Synchronous execution



Virtually synchronous execution

- **Virtual synchrony is a "consistency" model:**
  - ***Synchronous runs: indistinguishable from non-replicated object that saw the same updates (like Paxos)***
  - ***Virtually synchronous runs are indistinguishable from synchronous runs***

Example: Virtual Synchrony

- Virtual synchrony worked well with small numbers of groups, even at high event rates
  - As long as IPMC is exploited
- But most implementations are unstable:
  - With large numbers of groups
  - With extremely large groups
  - When membership churns with high multicast rate

Think of these as "properties"

- Simple best-effort reliability, but with a failure detection module
- Virtual synchrony
- Consensus-like protocols (Paxos)
- Byzantine Fault Tolerance
- Locking protocols (like Chubby)
- Transactional protocols (like Sinfonia)

Dimensions of scalability

- There may be many groups
- And there may be many pending multicasts
- And different uses may have different reliability requirements

- Today's protocols are often written as if there is just one group, and one flow of traffic.

Can we aggregate « properties »?

- Issue: today we tend to write protocols in Java or other explicit languages
- This interweaves "desired semantics" with implementation
  - The solution will scale exactly to the extent that the designer anticipated the need
  - Can we express "pure semantics" and later find ways to *generate* implementations that scale?

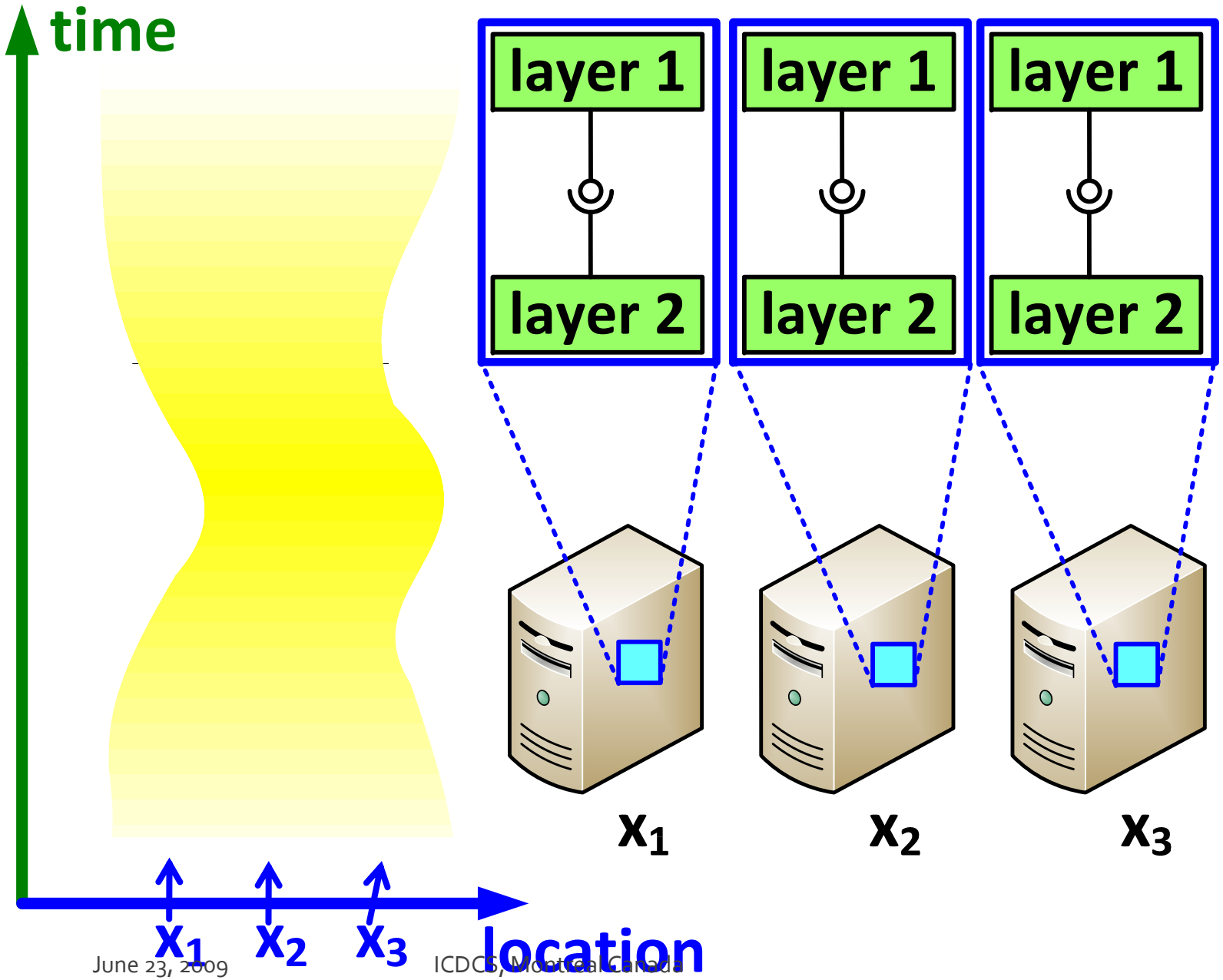
Ostrowski: Properties Framework

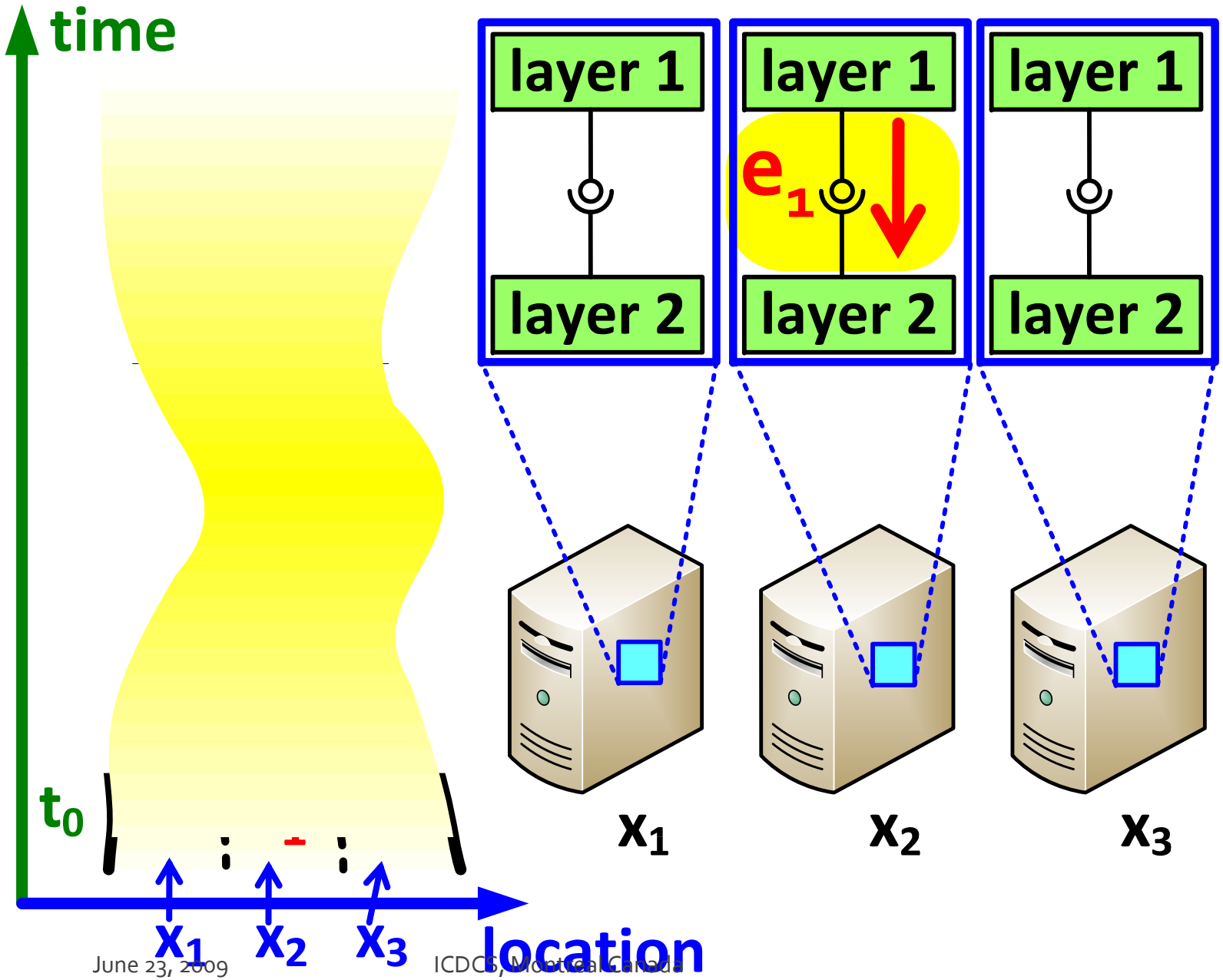
- Ostrowski is developing a properties "algebra" defined as predicates on "event flows"
  - Cleanly separates the property (say, all-or-none delivery, or total order) from implementation
  - Compiler generates actual runtime code with just-in-time knowledge of network topology, features
  - Optimizations are focused on "batching" operations
  - Quicksilver Multicast "interprets" the generated code!
- Yields protocols that perform well… and scale too

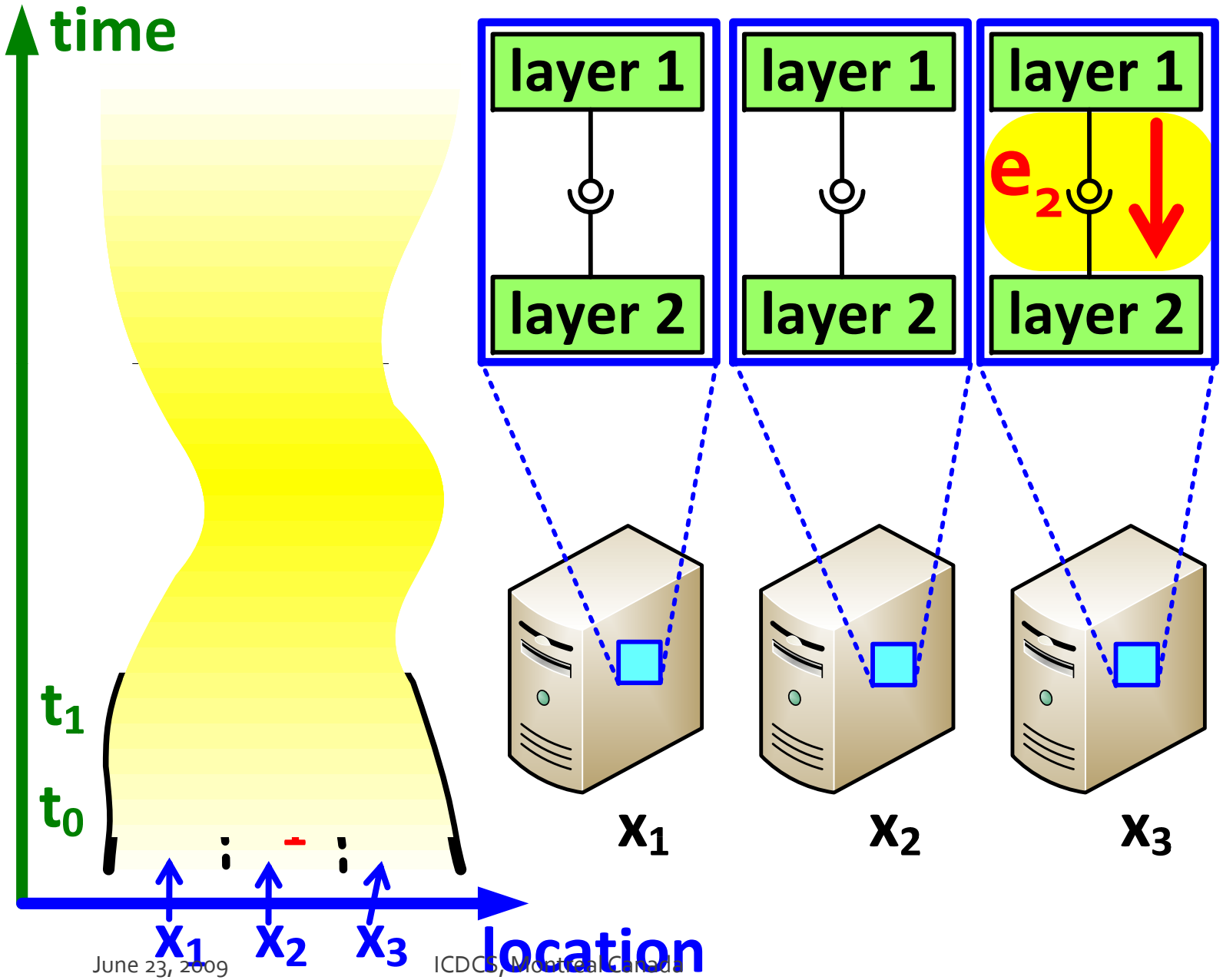
[DEBS 2009, ICWS 2009: Ostrowski et al]

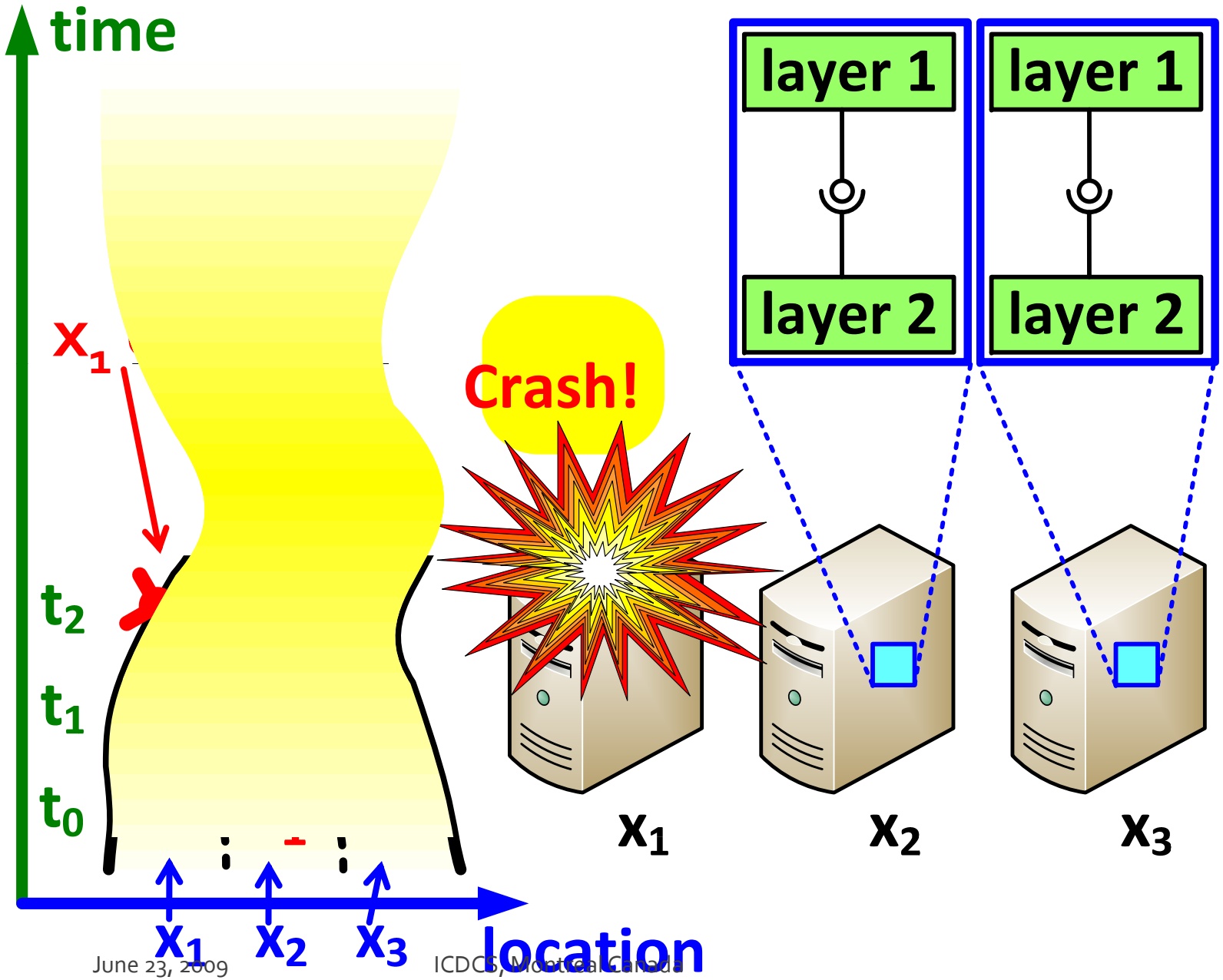
Goals?

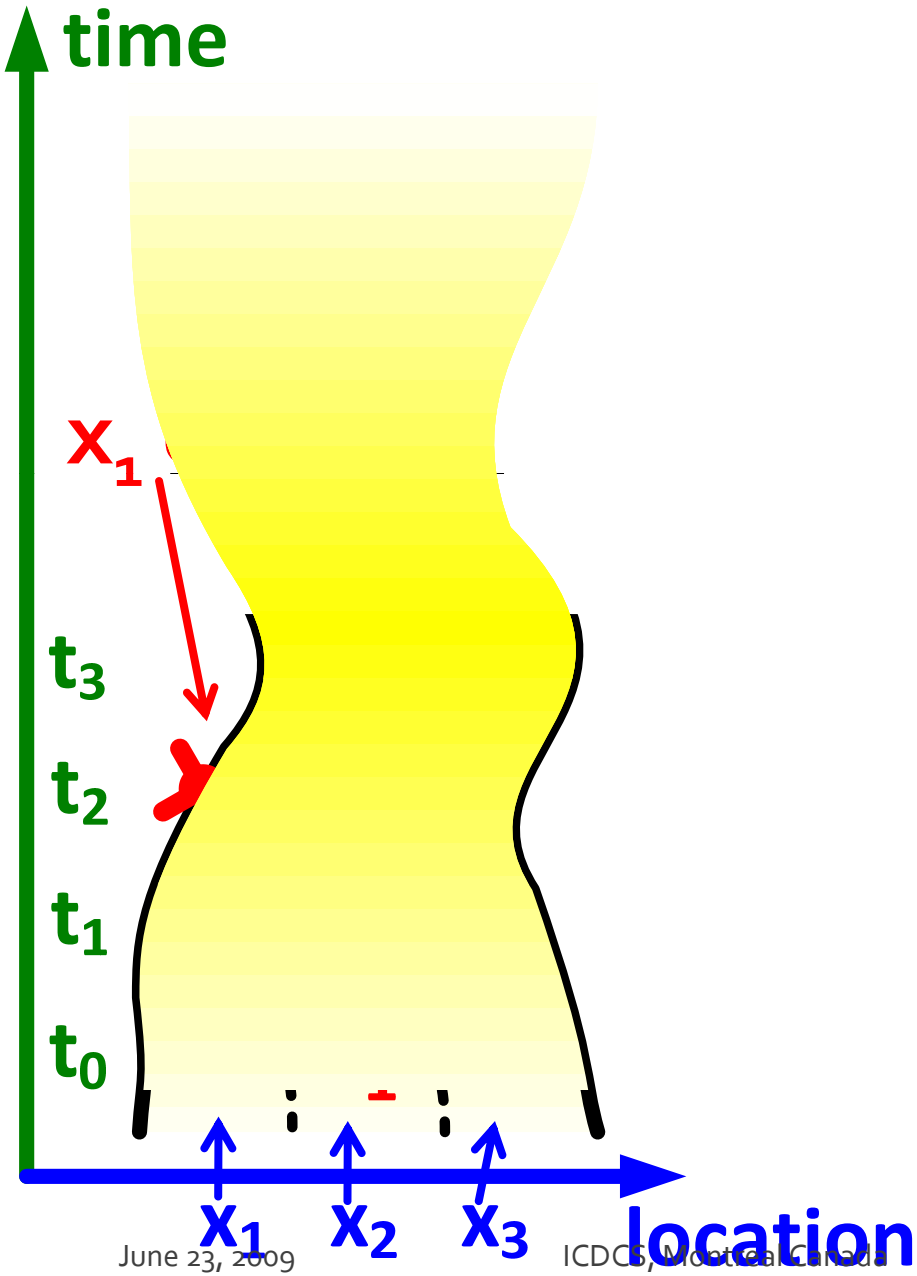
- **Speed:**
  - *Batch events to process huge numbers at a time for efficiency*
- **Size of groups:**
  - *IPMC to disseminate data (recall: now it almost never loses packets!)*
  - *Tree structures if group gets big*
- **Numbers of groups:**
  - *Try to combine "across" groups by factoring out shared properties. Feasible for simple reliability model used in QSM but seems hard for arbitrary properties*

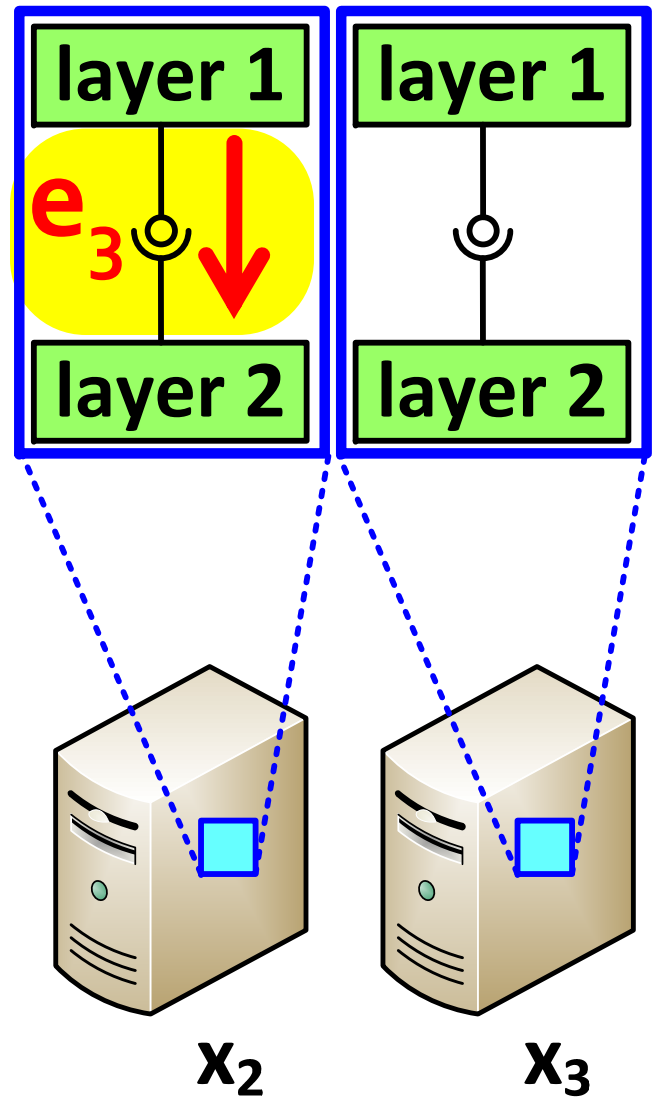

time
layer 1
layer 2
layer 1
layer 2
layer 1
layer 2
x_1
x_2
x_3
x_1
x_2
x_3
location

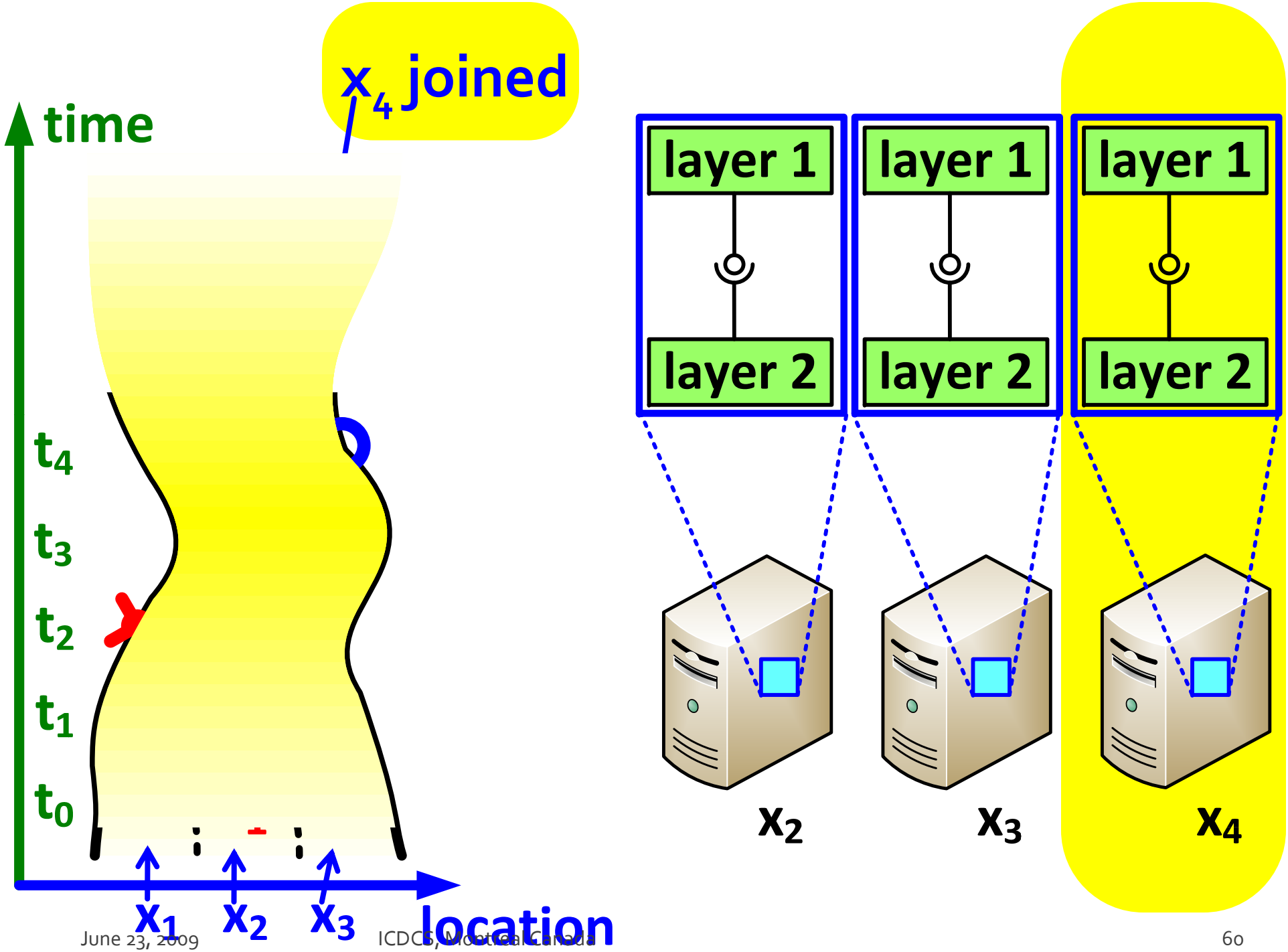

time
layer 1
layer 2
layer 1
e_1
layer 2
layer 1
layer 2
t_0
x_1
x_2
x_3
x_1
x_2
x_3
location

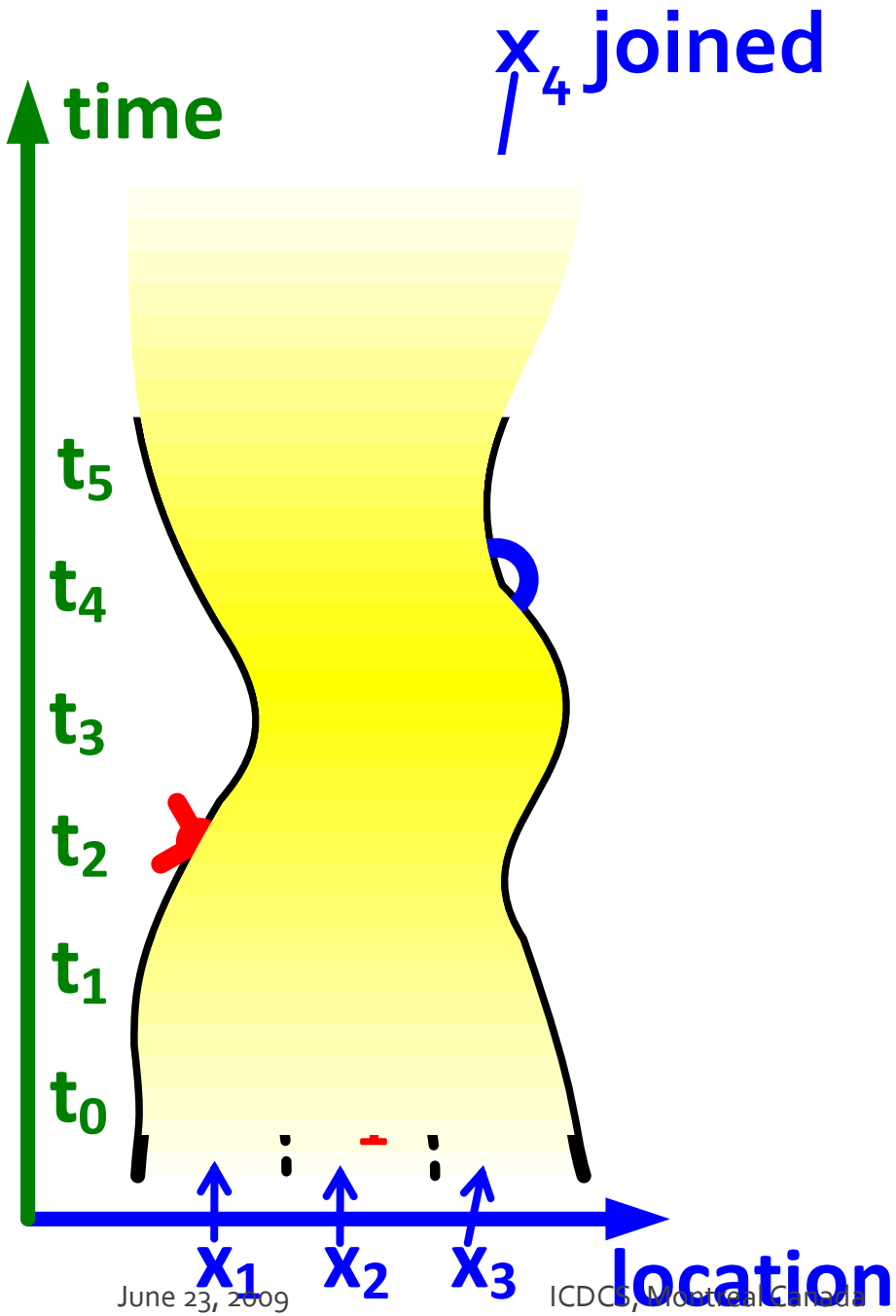

time
layer 1
layer 2
layer 1
layer 2
layer 1
e_2
layer 2
t_1
t_0
x_1
x_2
x_3
x_1
x_2
x_3
location

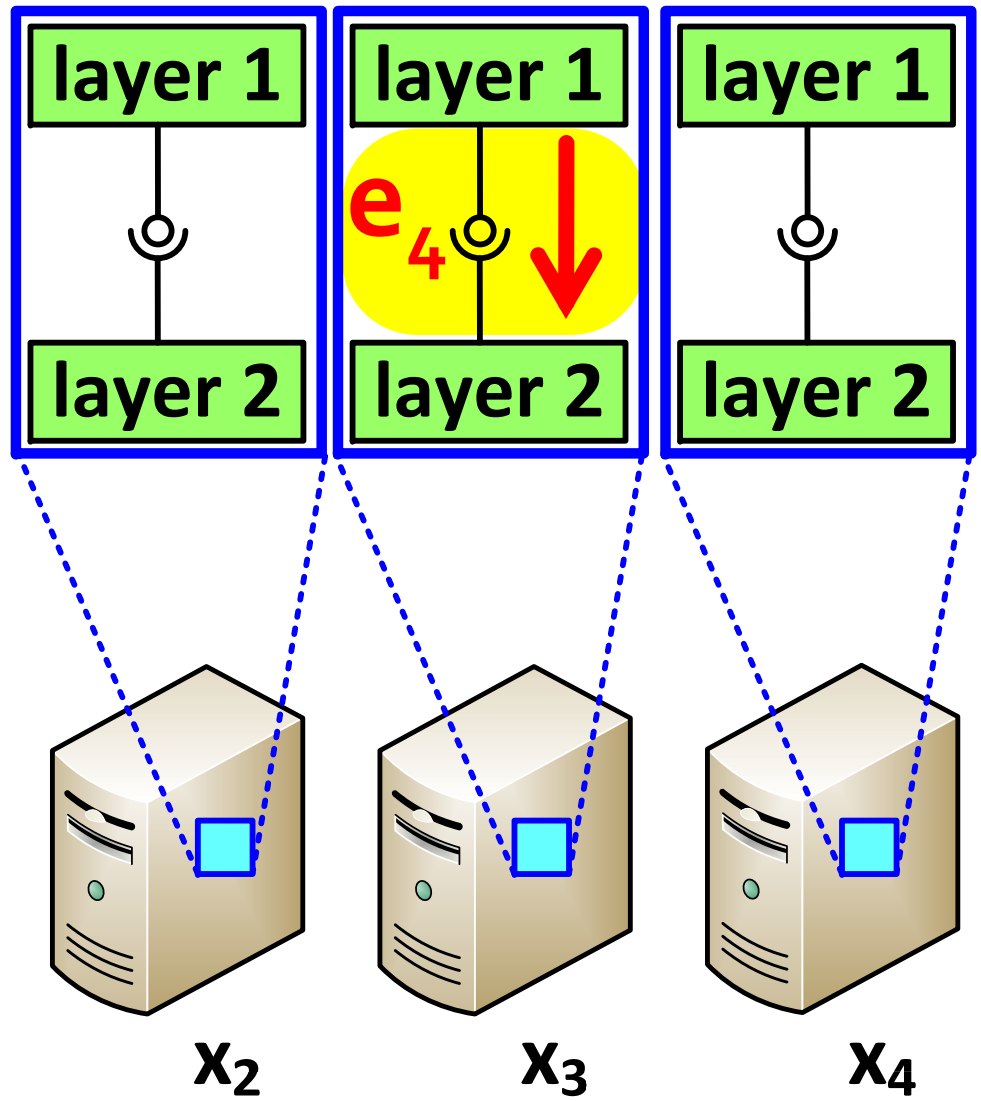

time
x_1
t_2
t_1
t_0
x_1
x_2
x_3
location
Crash!
layer 1
layer 2
layer 1
layer 2
x_1
x_2
x_3

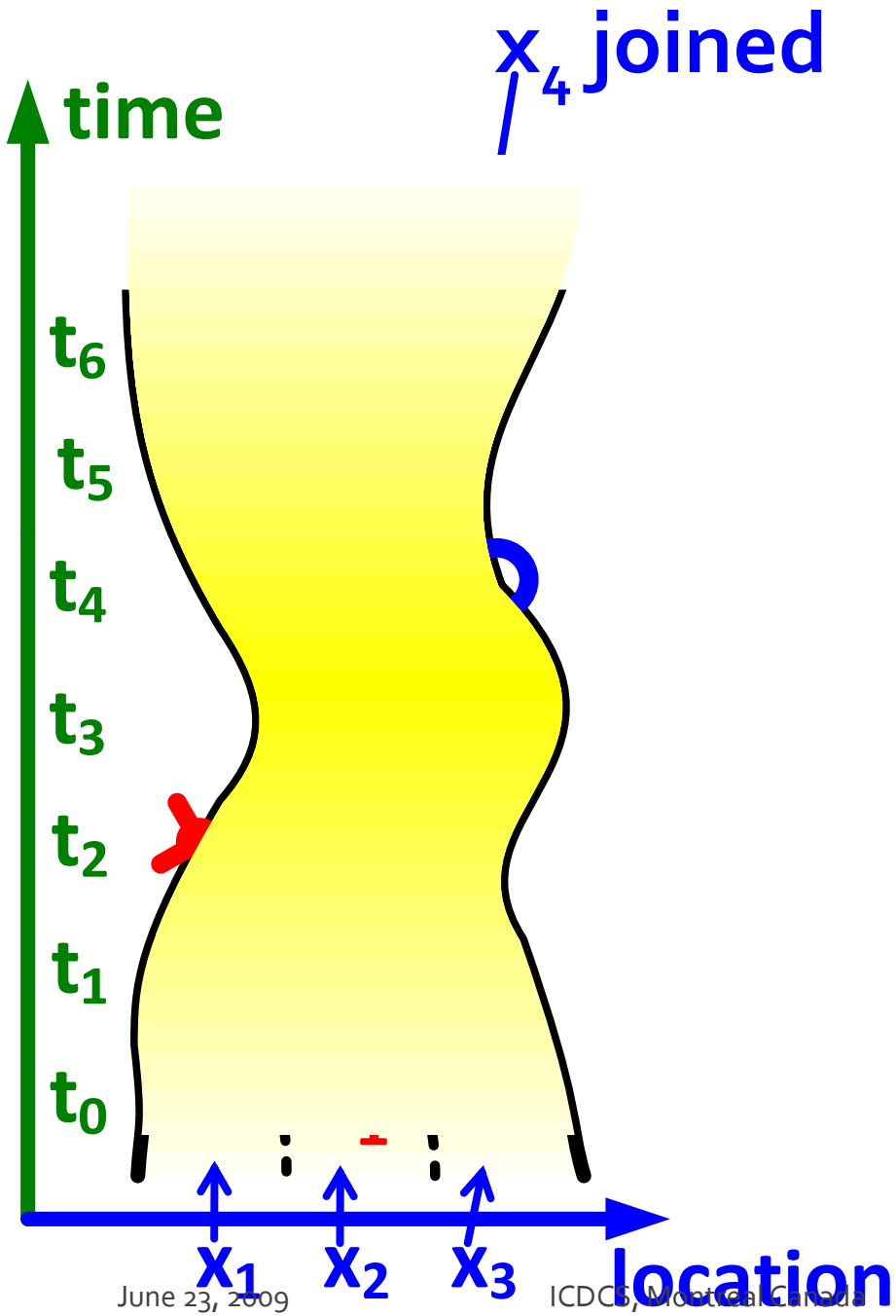

time
x_1
t_3
t_2
t_1
t_0
x_1
x_2
x_3
location

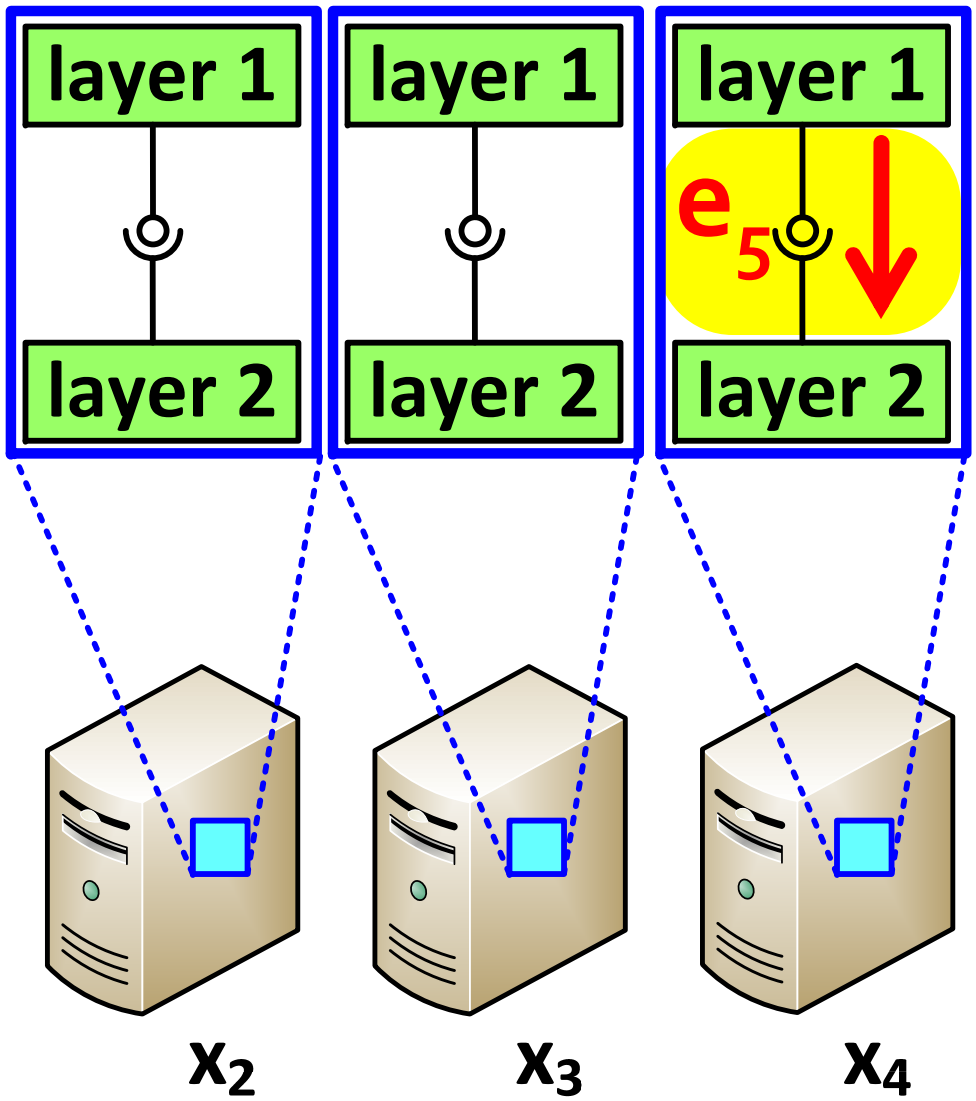

layer 1
layer 2
e_3
layer 1
layer 2
x_2
x_3

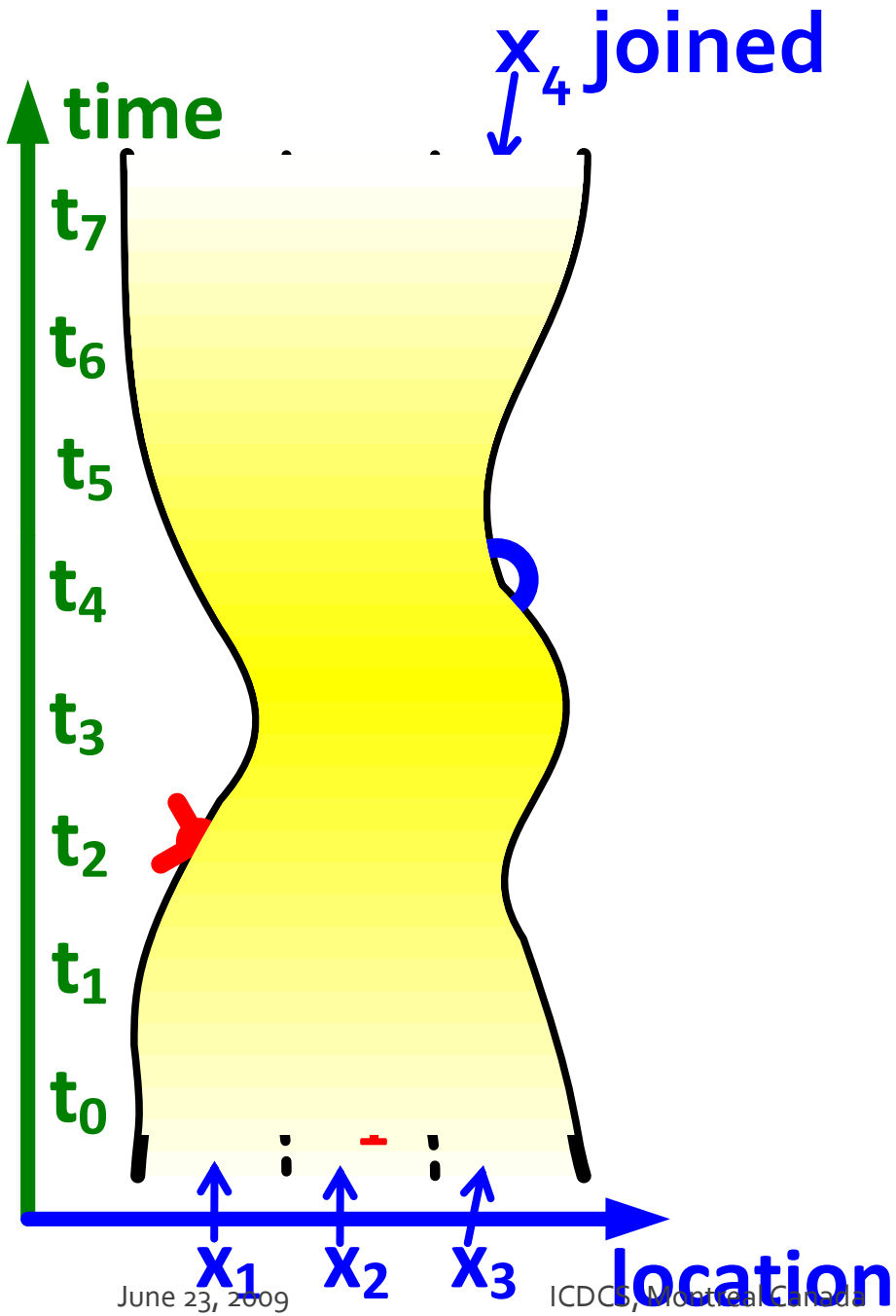

x_4 joined
time
t_4
t_3
t_2
t_1
t_0
x_1
x_2
x_3
location
layer 1
layer 2
layer 1
layer 2
layer 1
layer 2
x_2
x_3
x_4

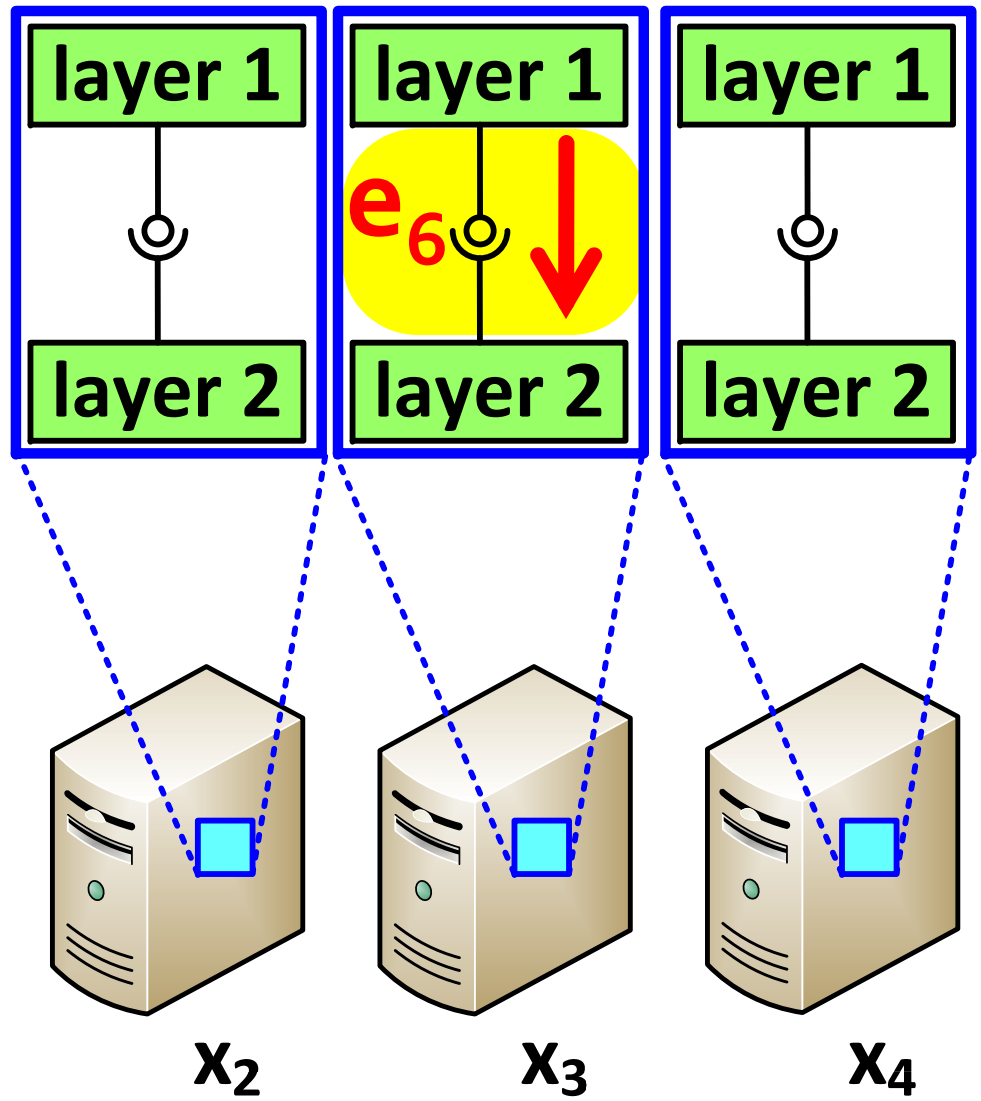

x_4 joined
time
t_5
t_4
t_3
t_2
t_1
t_0
x_1
x_2
x_3
location

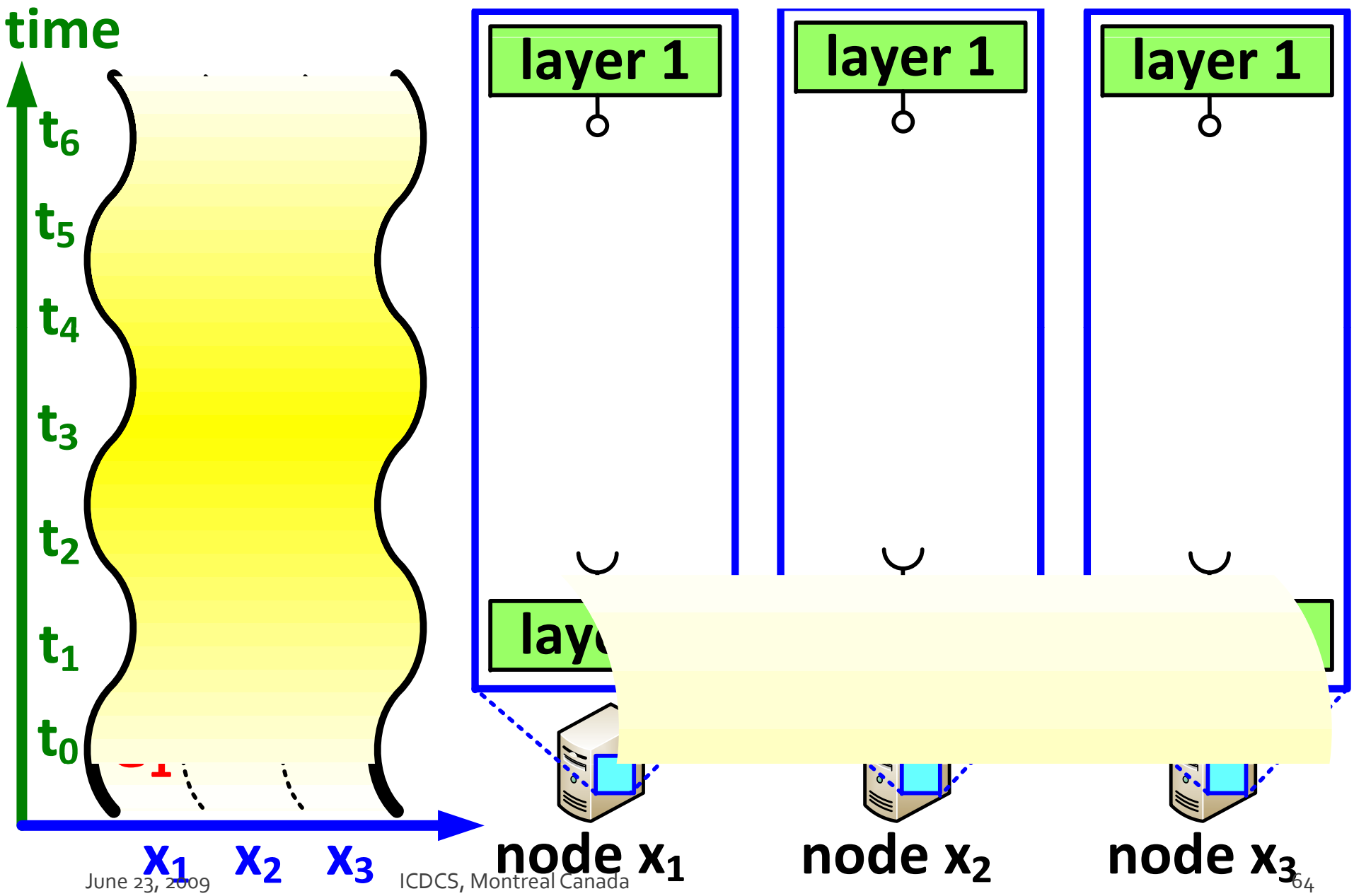

layer 1
layer 2
layer 1
e_4
layer 2
layer 1
layer 2
x_2
x_3
x_4


x_4 joined
time
t_6
t_5
t_4
t_3
t_2
t_1
t_0
x_1
x_2
x_3
location

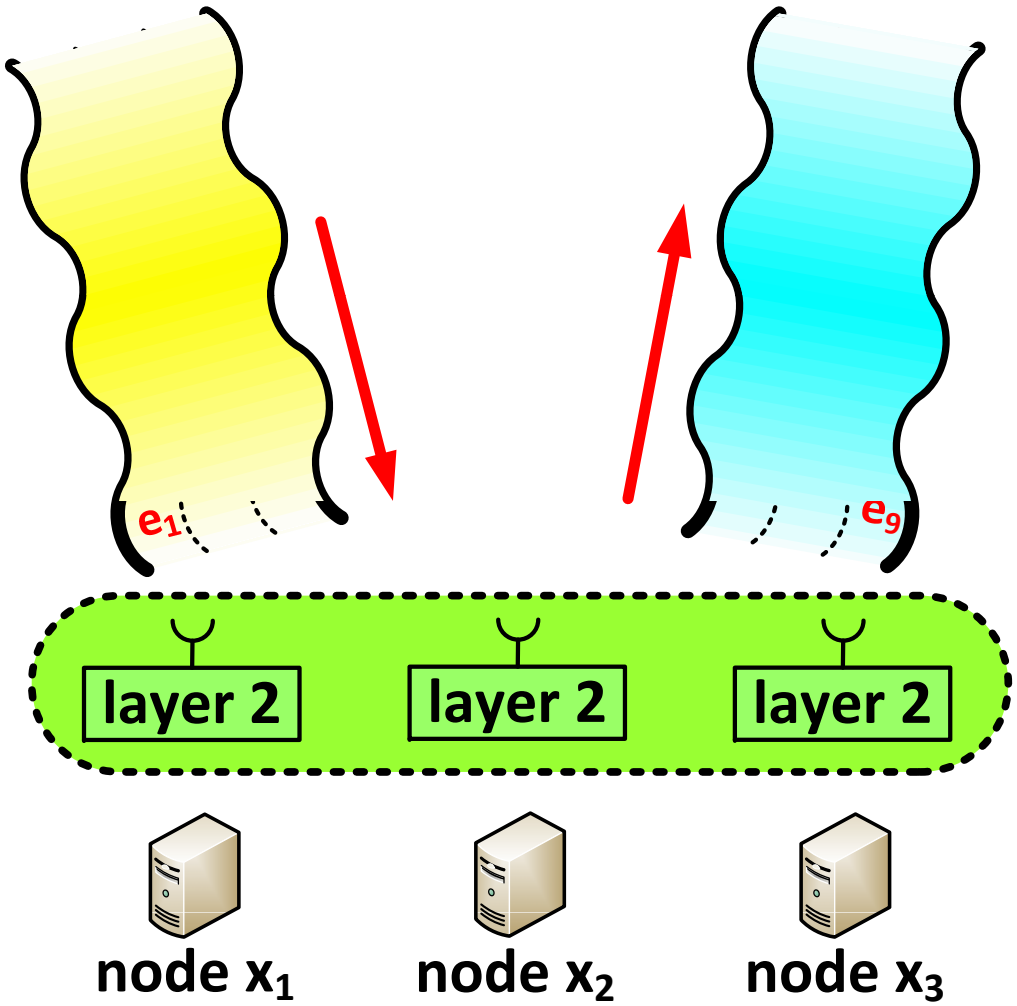

layer 1
layer 2
layer 1
layer 2
layer 1
e_5
layer 2
x_2
x_3
x_4


x_4 joined
time
t_7
t_6
t_5
t_4
t_3
t_2
t_1
t_0
x_1
x_2
x_3
location

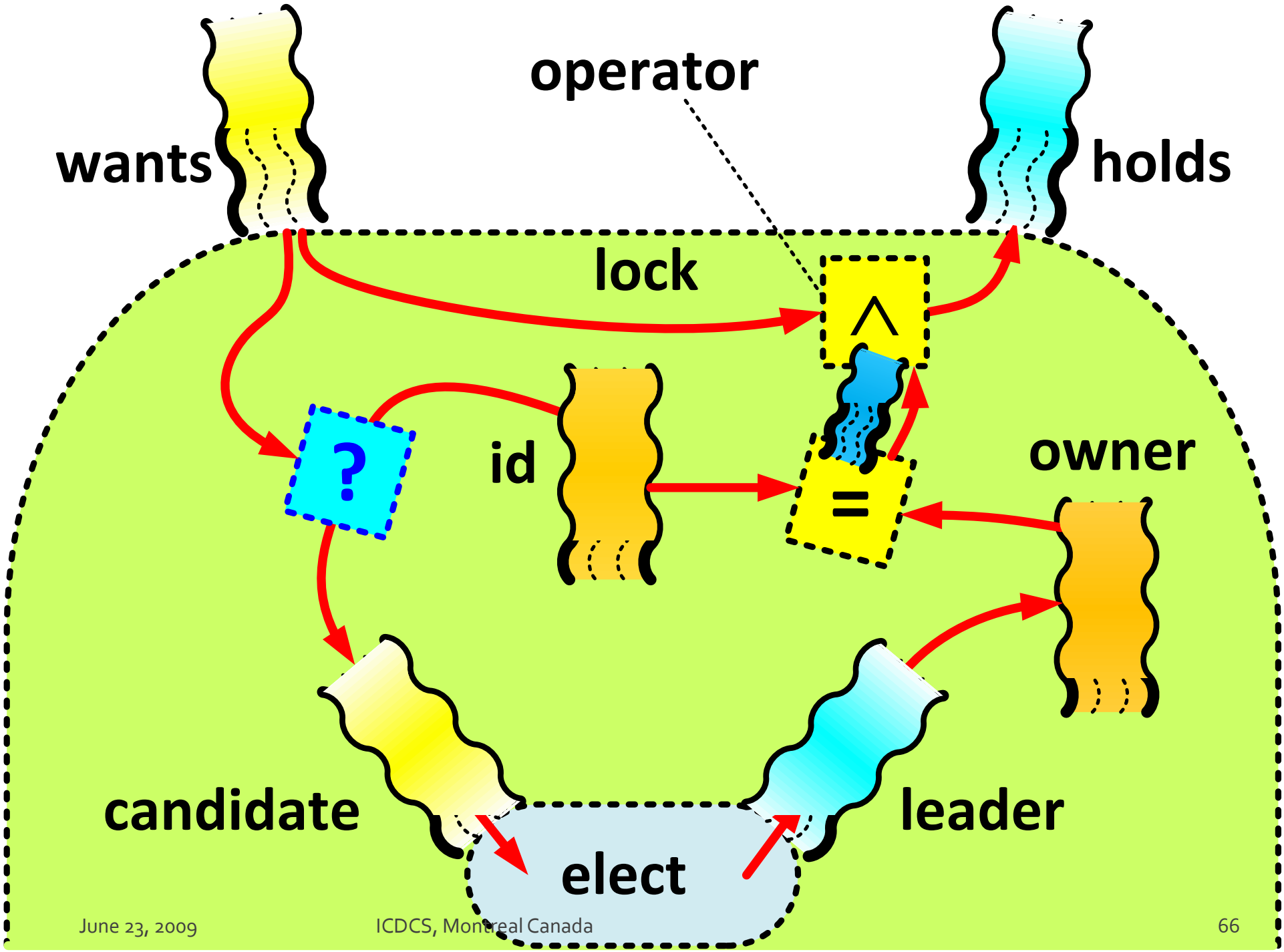

layer 1
layer 2
layer 1
e_6
layer 2
layer 1
layer 2
x_2
x_3
x_4


time
t_6
t_5
t_4
t_3
t_2
t_1
t_0
x_1
x_2
x_3
layer 1
layer 1
layer 1
node x_1
node x_2
node x_3

Ostrowski: Properties Framework



[OOPSLA 2009, DEBS 2009, ICWS 2009: Ostrowski et al]


operator
wants
holds
lock
^
?
id
=
owner
candidate
leader
elect

Ostrowski: Properties Framework

```
object select(up int candidate) : s-up int leader
{
    s-up int selected := 0;
    where (fresh selected ∧ selected ≤ candidate)
        selected := min candidate;
    leader := selected;
}
```

[DEBS 2009, ICWS 2009: Ostrowski et al]

Role of Compiler

- Implements a bag of "scalable tricks"
  - Operates on batches of events
  - Uses efficient representations for things like sequence numbers and sets
  - As asynchronous as possible
  - Employs hierarchical structures that scale well (tree of token rings)
- To date, have applied to simple reliability properties and virtual synchrony

Before/After Properties Framework

- Traditional protocols are "overspecified"
  - They conflate *implementation* with *specification*
  - Properties Framework specs are pure semantics: implementation left to compiler, runtime system
- Lets us automate scalability
  - … in size of groups (via hierarchy).
  - … over multicast flows (by aggregating).
  - For simple reliability properties, over multiple groups

Summary?

- CTOs hate consistency because
  - Consistency mechanisms are associated with disruptive network meltdowns
  - Anyhow, existing protocols don't scale well

- Dr Multicast can cure your network woes
- Properties Framework automates scalability

Closing: A Challenge…

- Today's embrace of inconsistency has given us scalable services we just can't trust.
- As cloud takes on medical care, banking, critical tasks… inconsistency just isn't good enough
- The new agenda: *properties without fear*

http://www.cs.cornell.edu/ken/ has papers on our work.