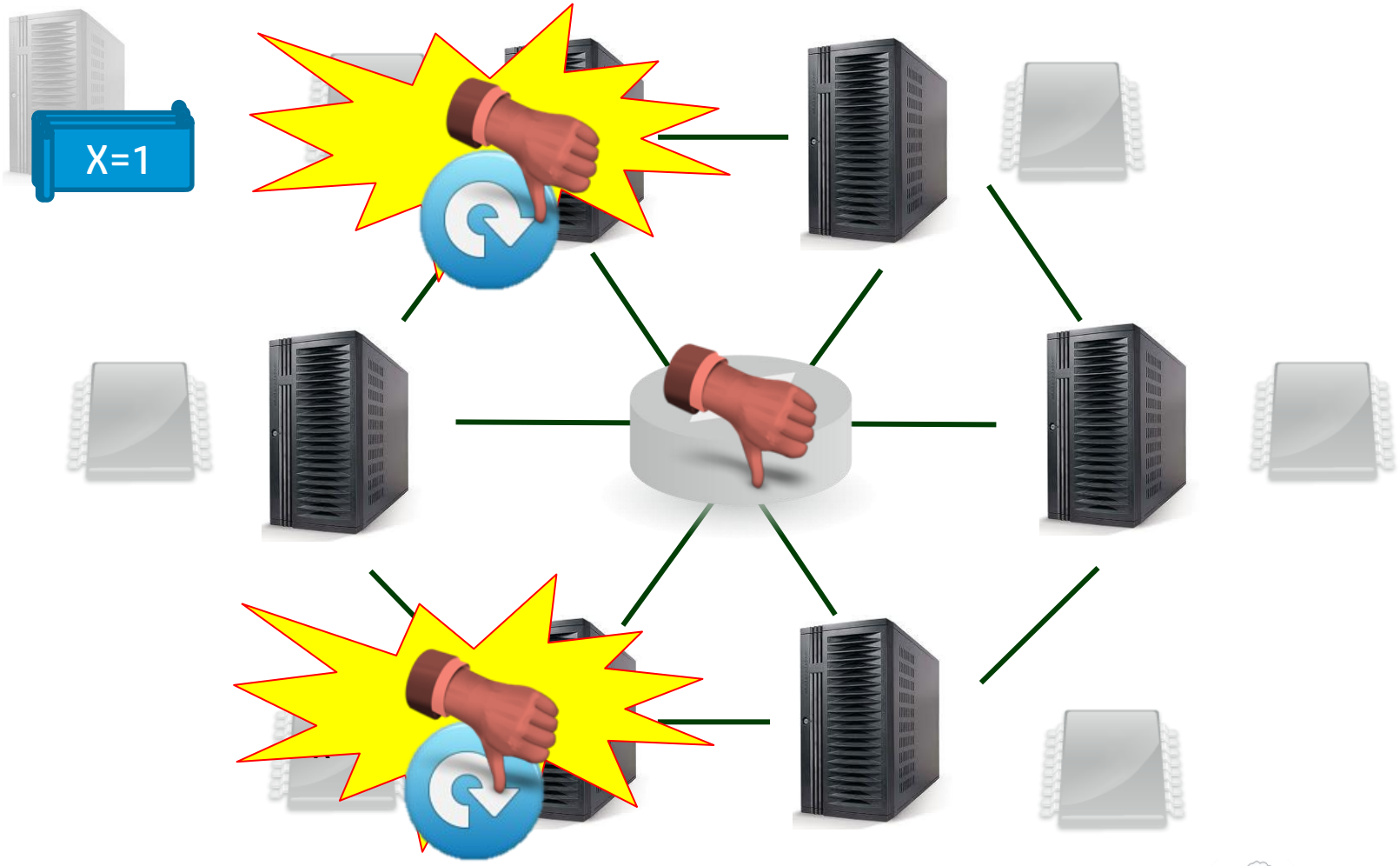





Composable Reliability for Asynchronous Systems

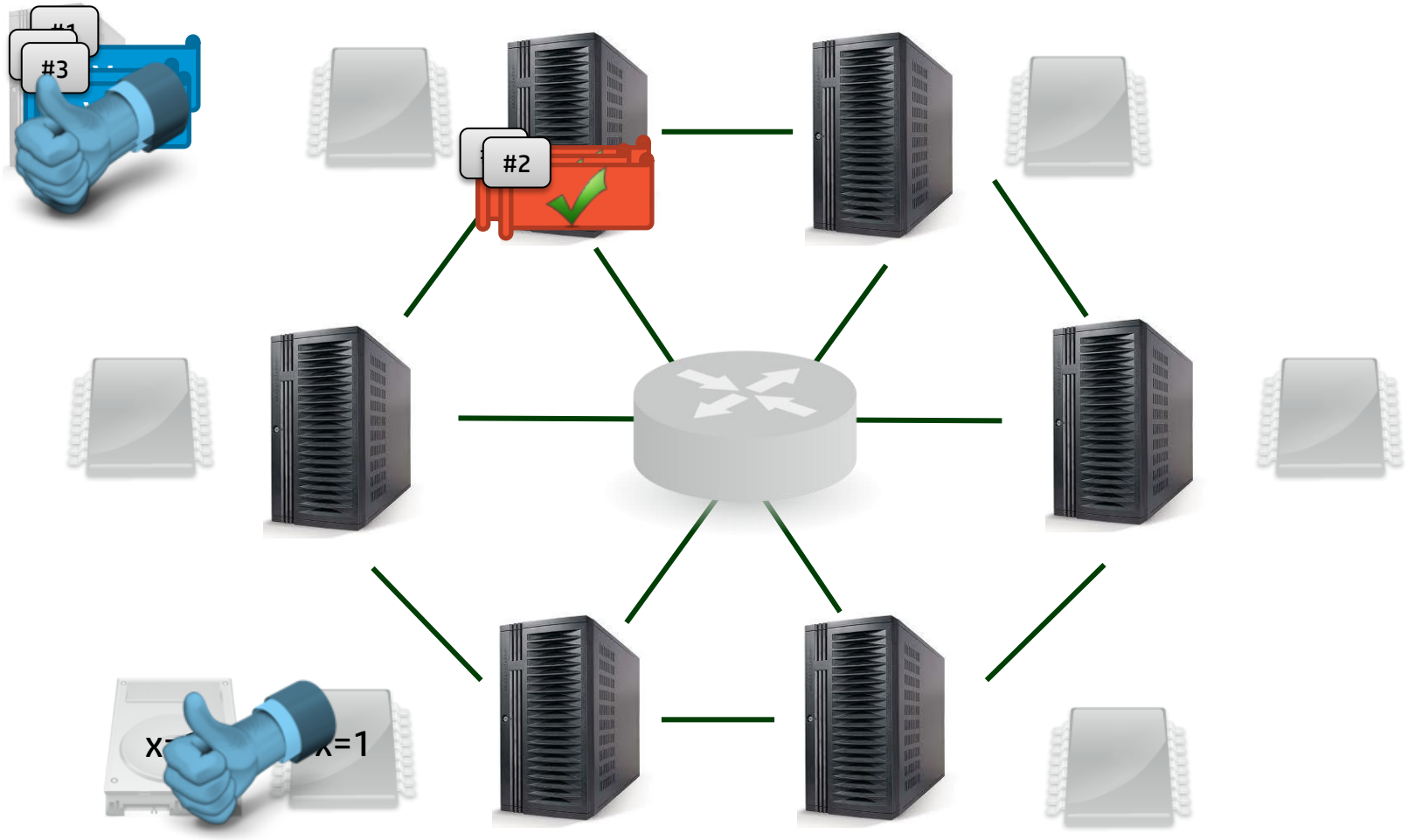
Sunghwan Yoo, Charles Killian, Terence Kelly,
Hyoun Kyu Cho, Steven Plite

Distributed systems: Key-value store



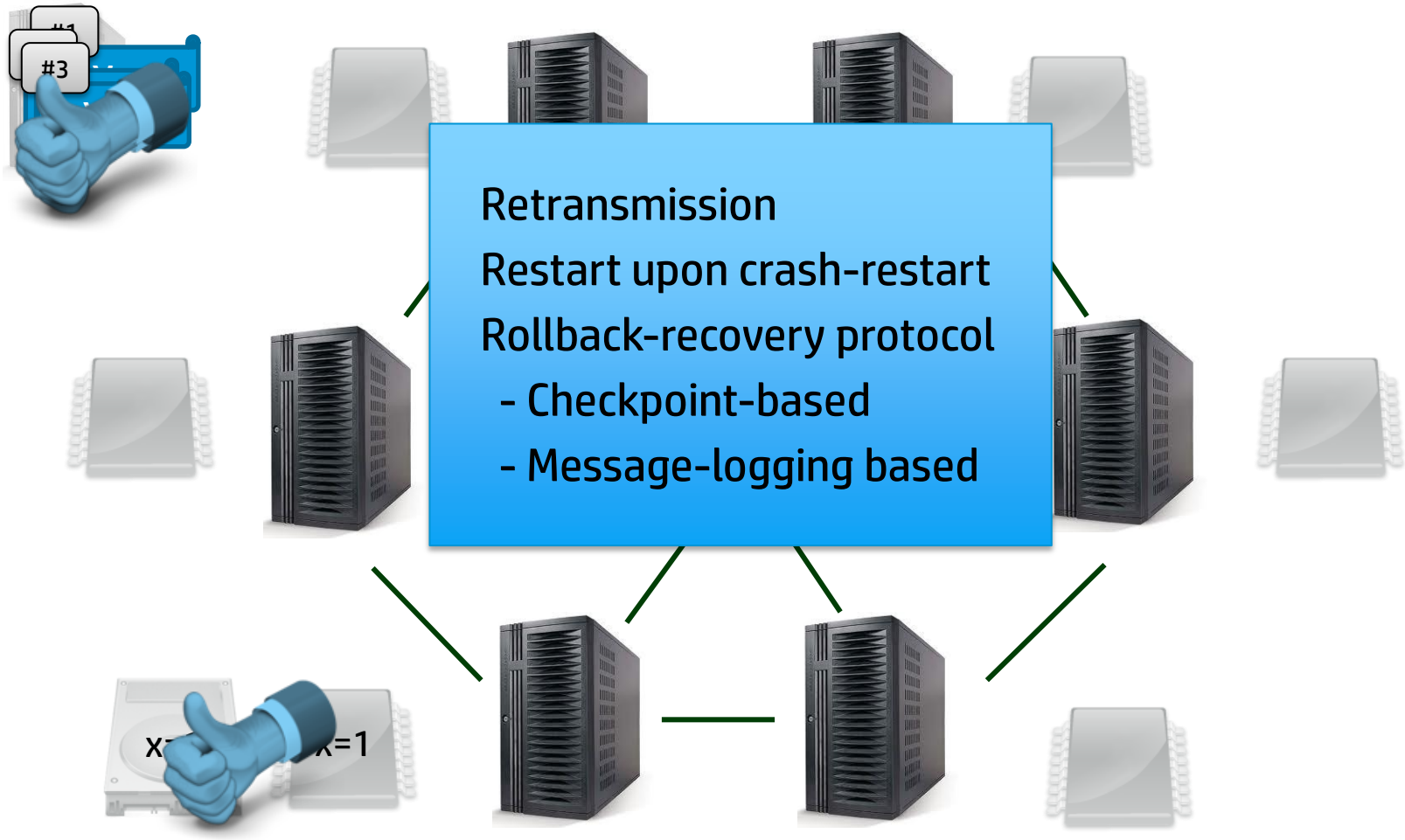
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Distributed systems: Key-value store



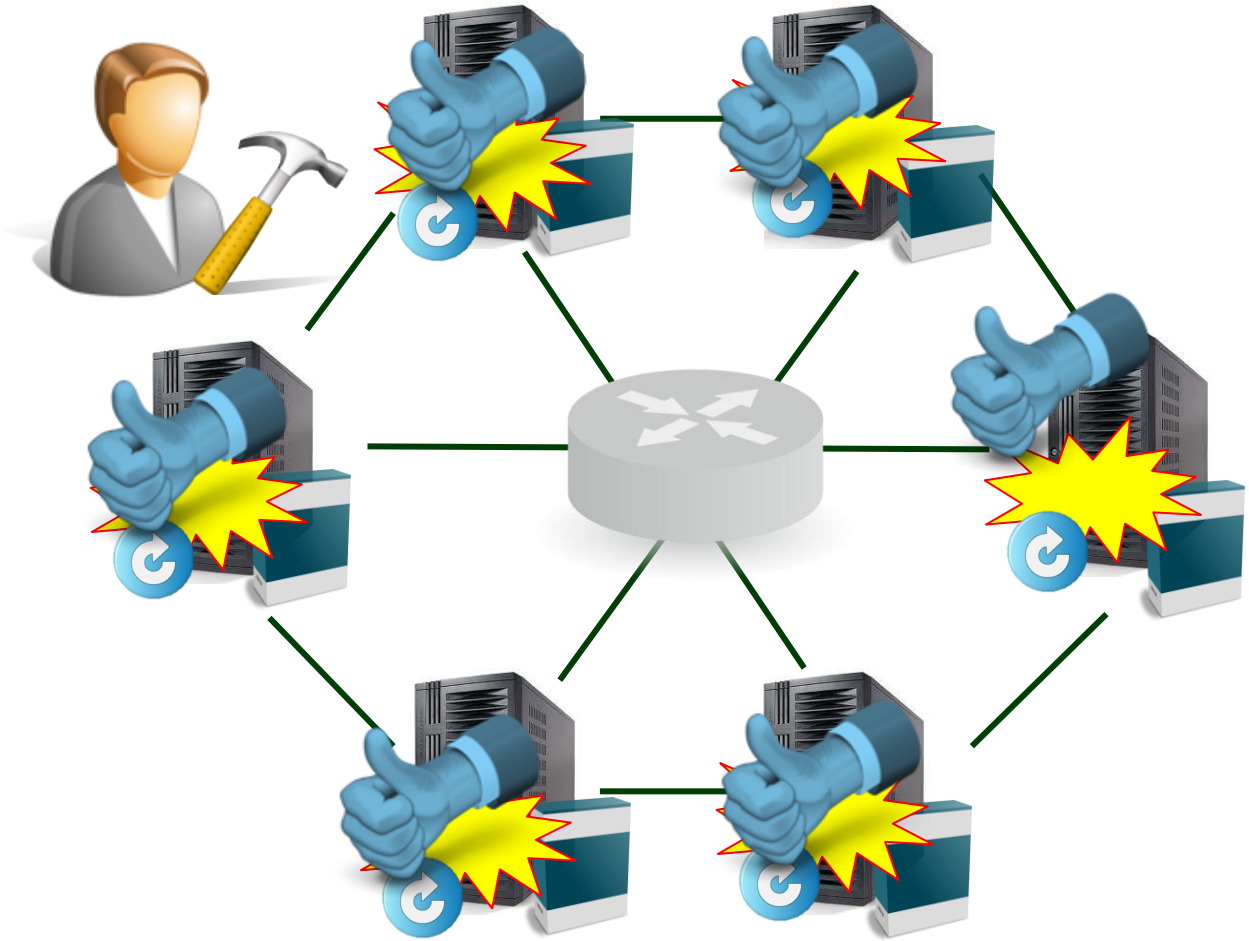
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Distributed systems: Key-value store



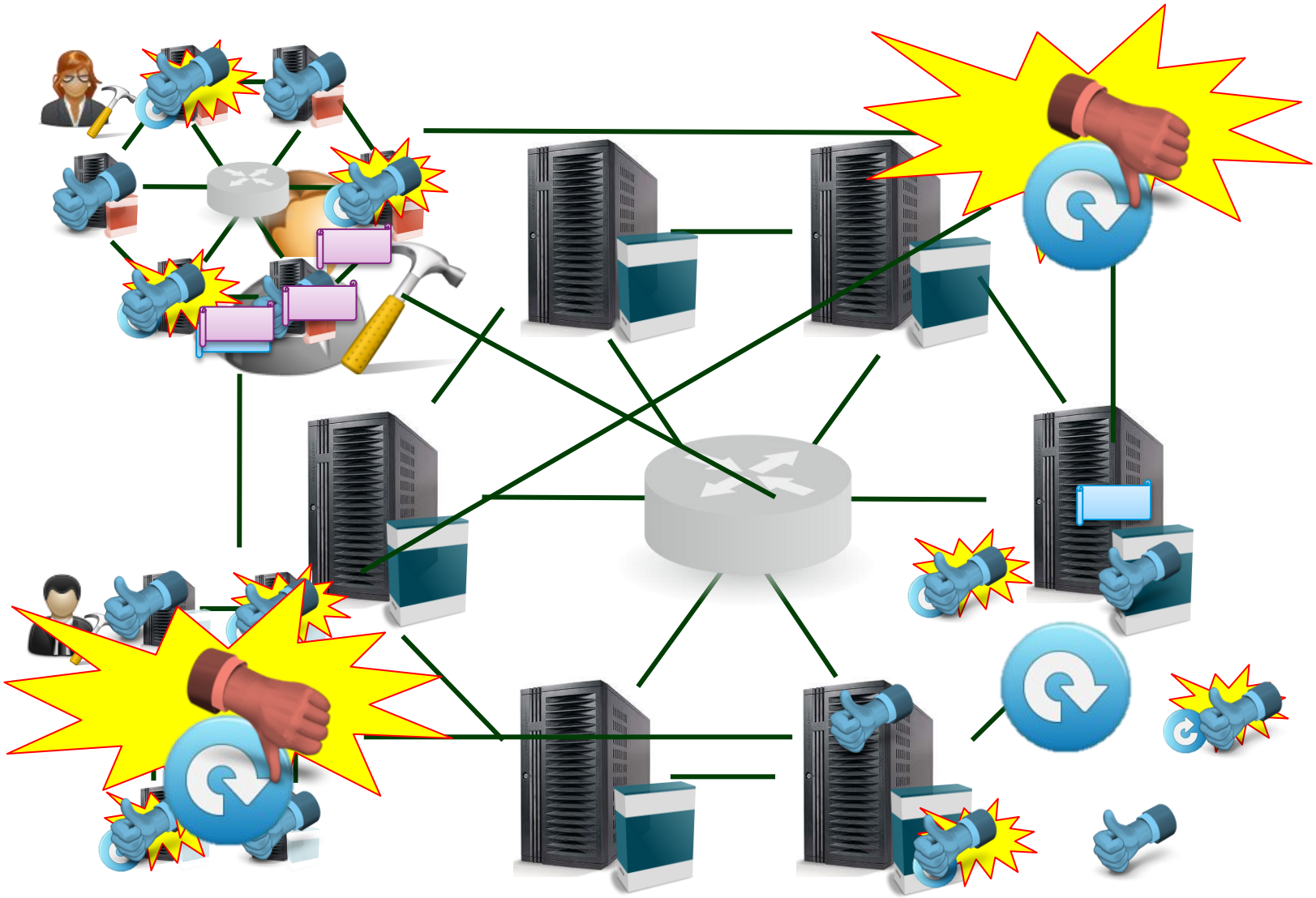
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Distributed systems: Handling failures



Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Distributed systems: Handling failures



Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

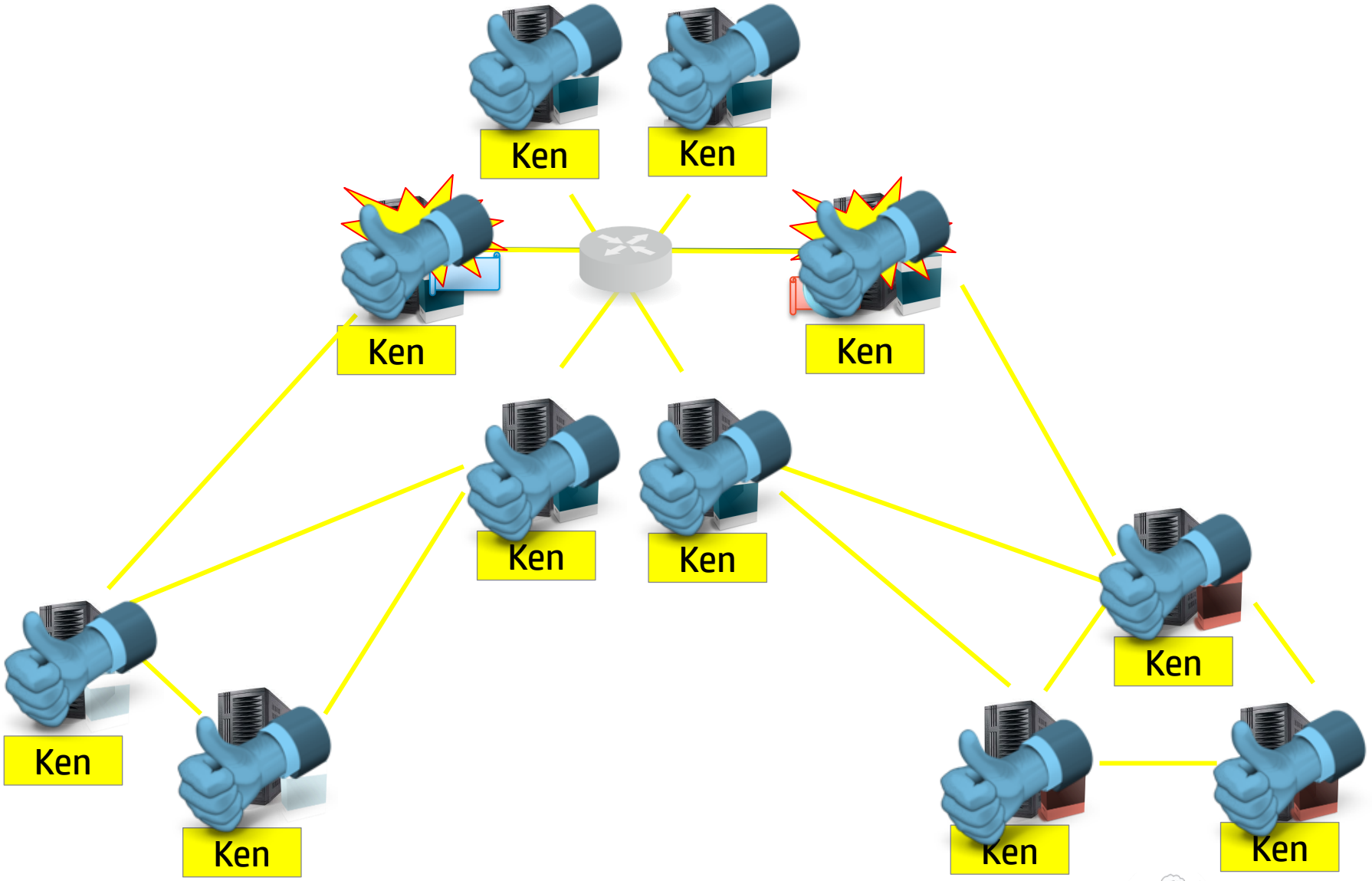
Distributed systems: Handling failures



Guaranteeing global reliability across independently developed components is difficult.

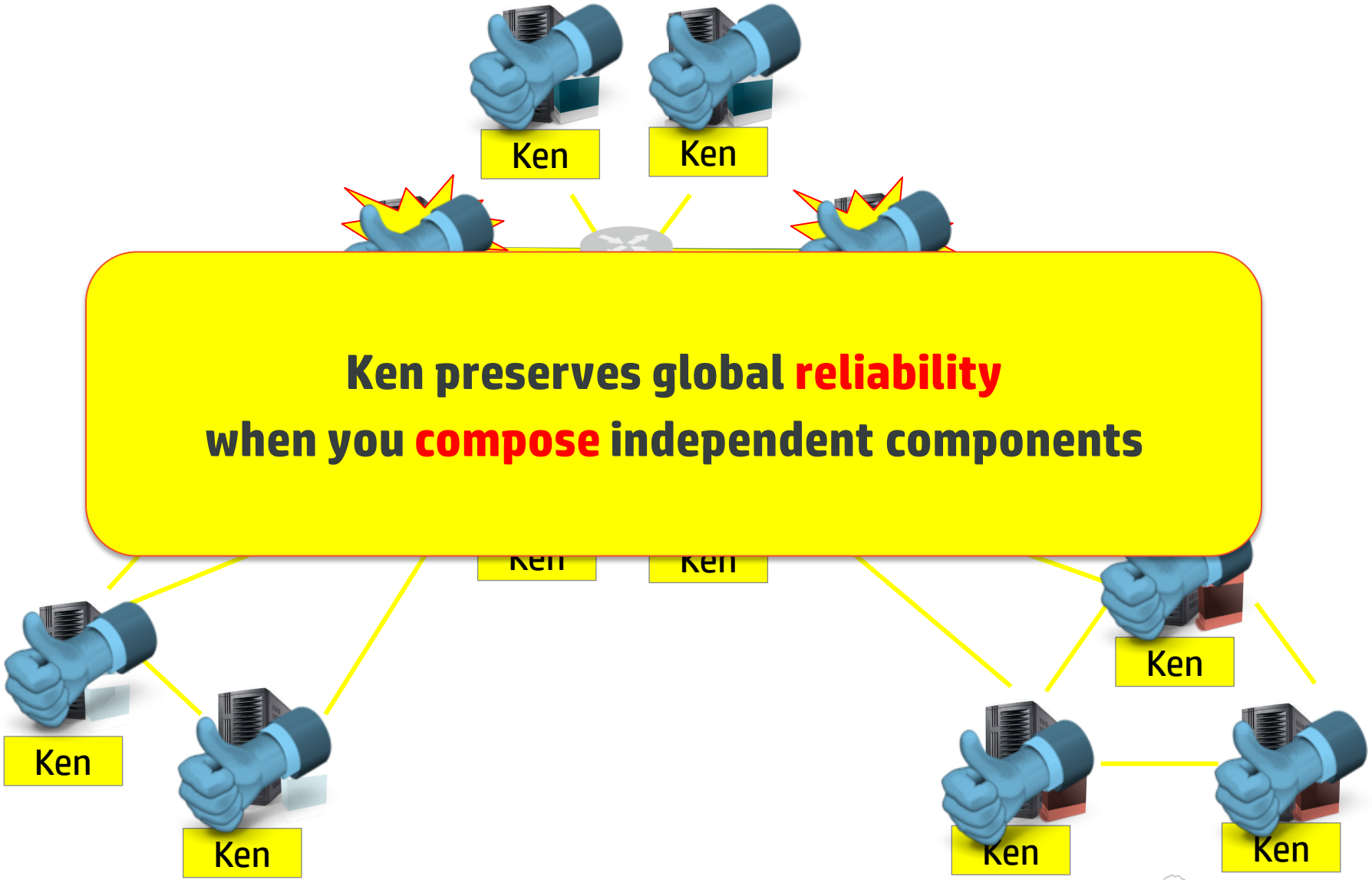
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Ken: Crash-restart tolerant protocol



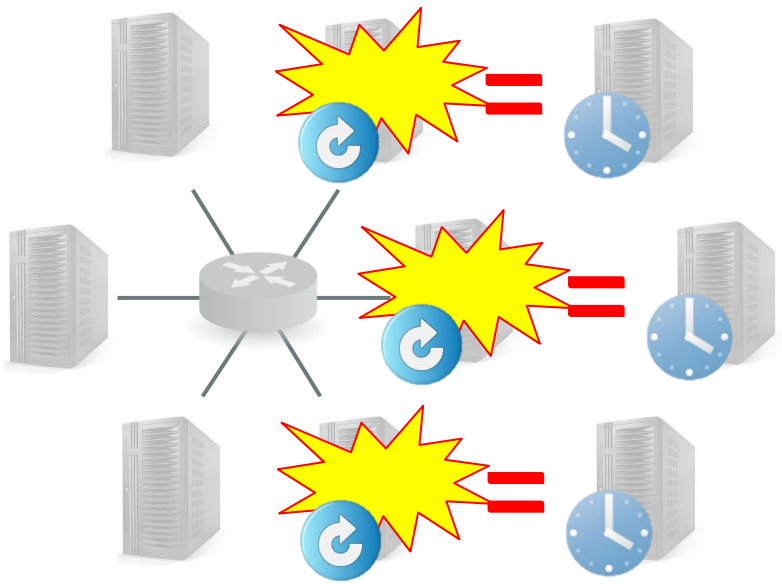
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Ken: Crash-restart tolerant protocol



Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Ken Highlights



Ken
makes crash-restarted node
look like slow node

Ken provides

1. Reliability
   - ❑ Uncoordinated rollback recovery protocol
   - ❑ It's also scalable
2. Composability
   - ❑ Write locally, work globally
3. Easy Programmability
   - ❑ Event-driven system (not a new paradigm)
   - ❑ Transparently applicable to Mace

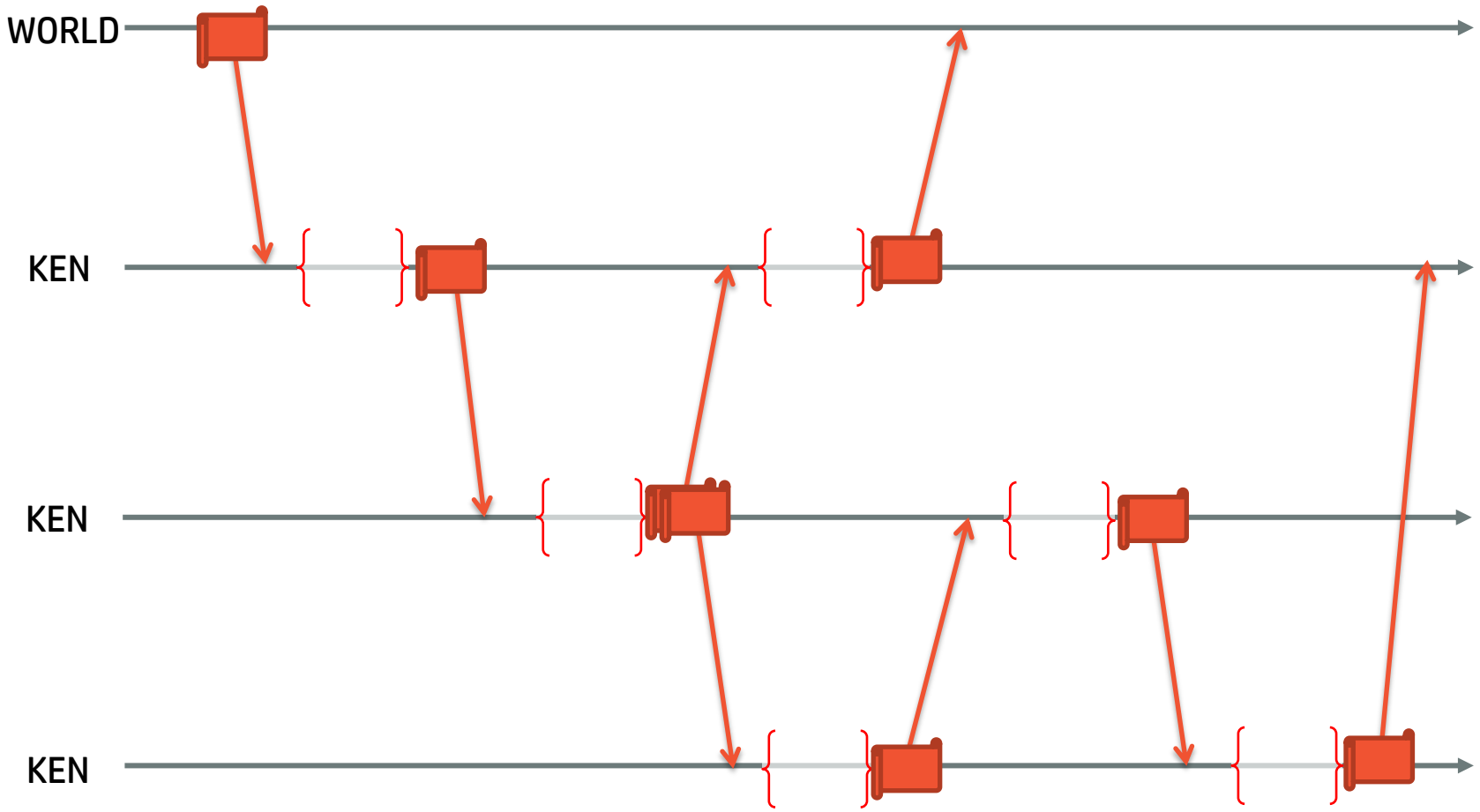
Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Related Work

- Rollback Recovery
  - Much research through 1990s
  - Waterken (1999) : Ken principles in different programming abstractions
  - Lowell et al. (2000) : Output validity
  - Computing Surveys (2002): summary of mature field
- Software Persistent Memory (ATC 2012)
  - Different approach to orthogonal persistence
- Hardening Crash-Tolerant Systems (ATC 2012)
  - Detects arbitrary state corruption in event-driven code
  - Could make Ken-based software more reliable

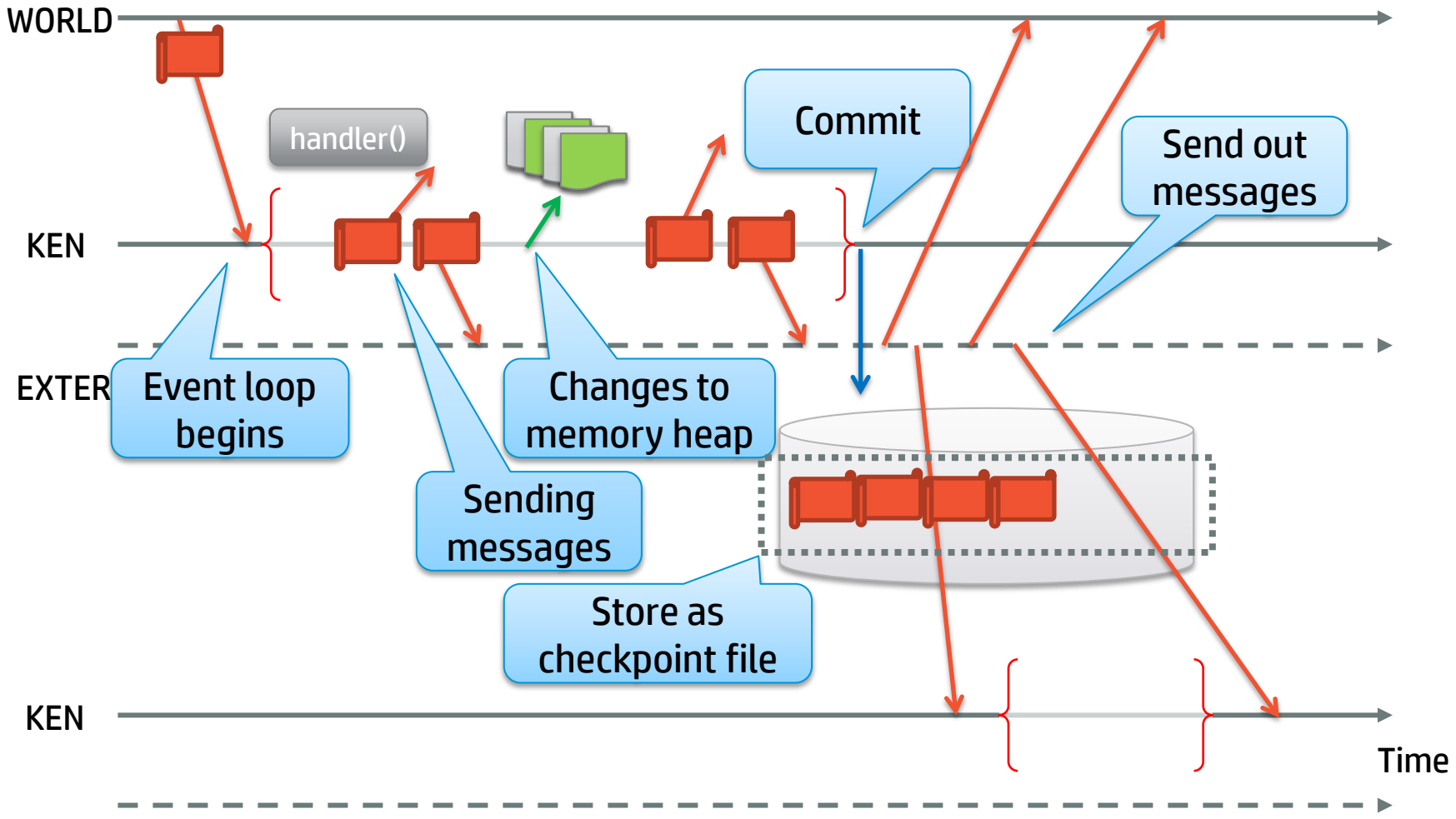
Design of Ken

- Communicating event loop



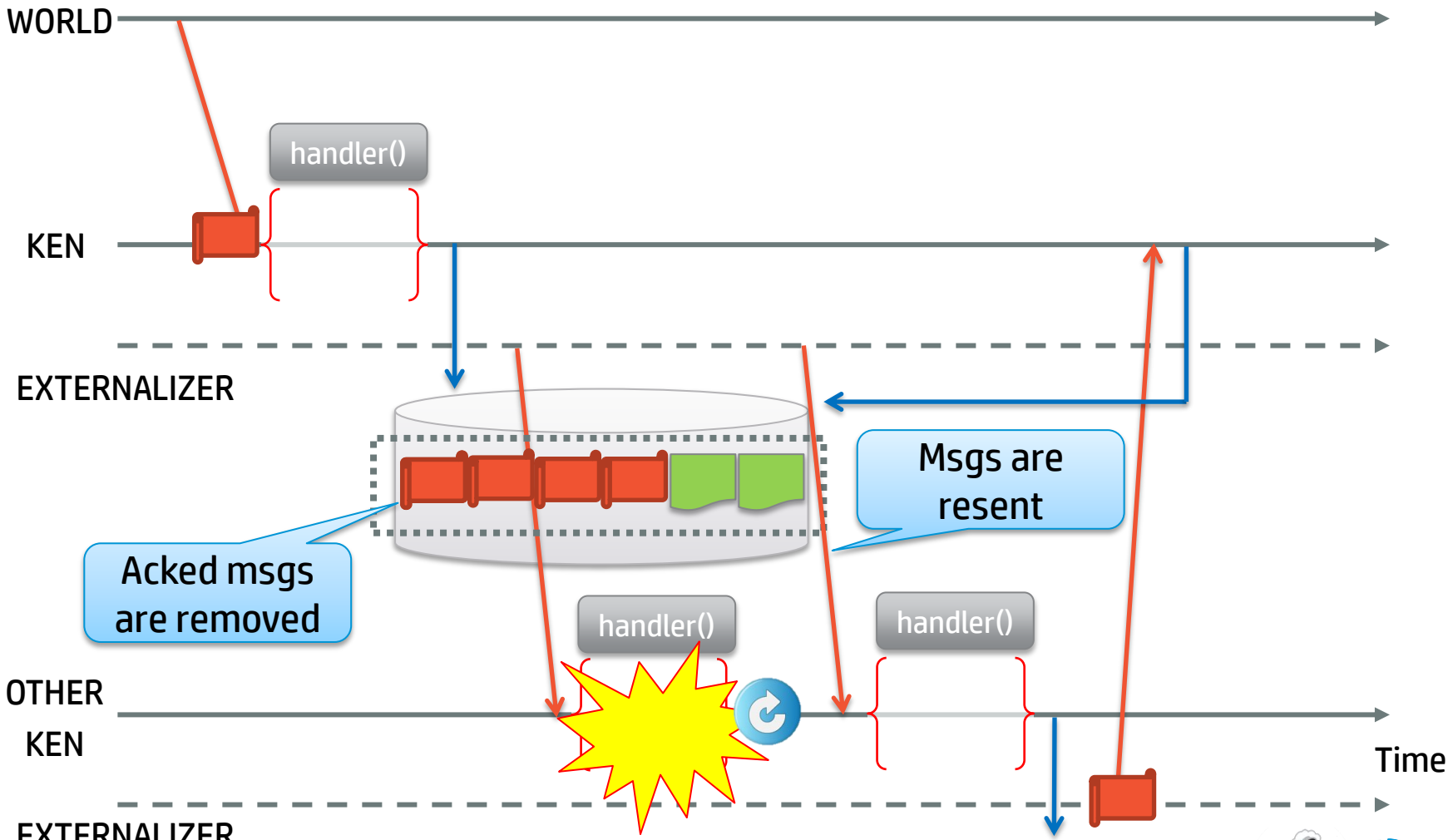
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Design of Ken



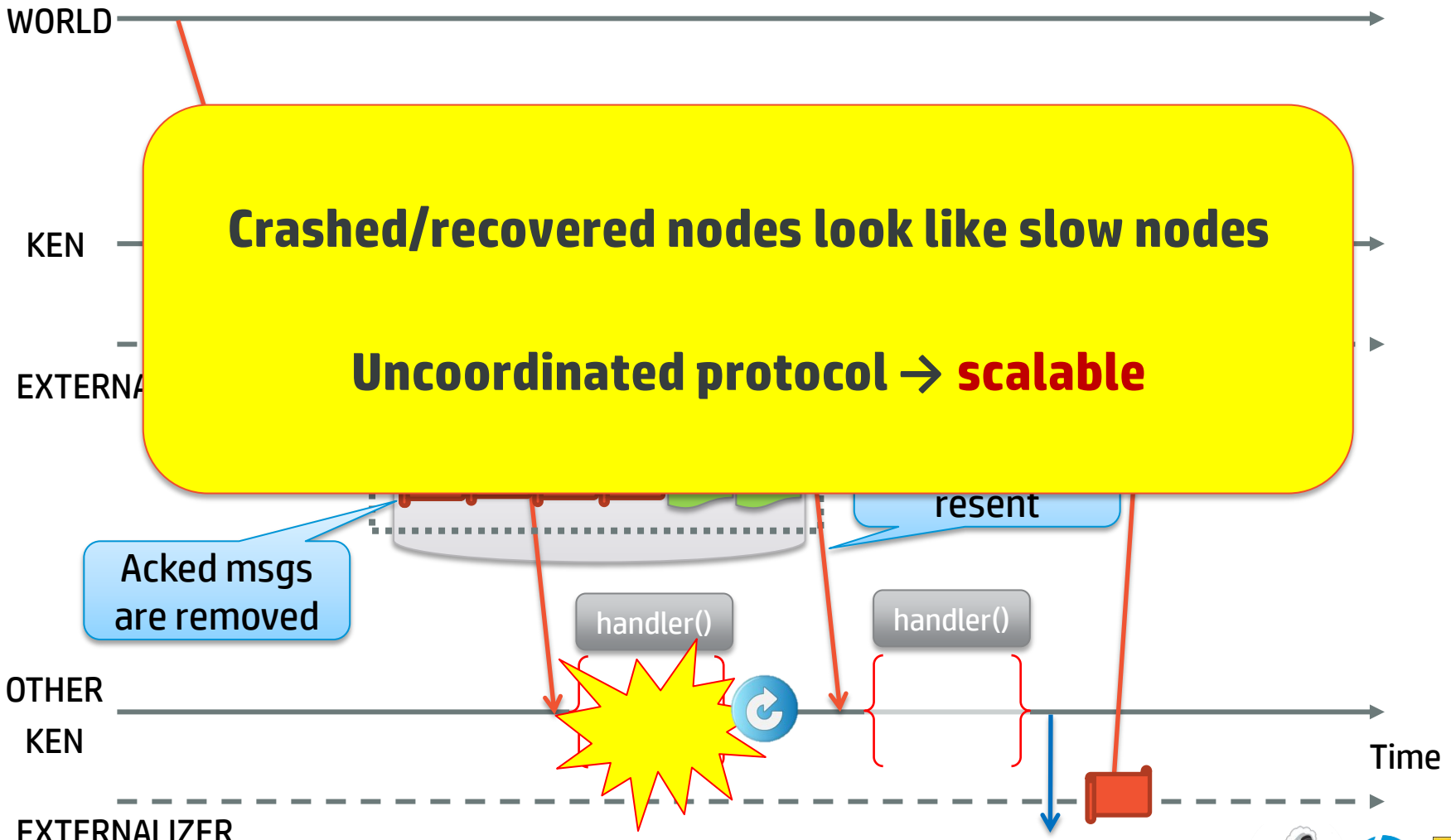
Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Design of Ken



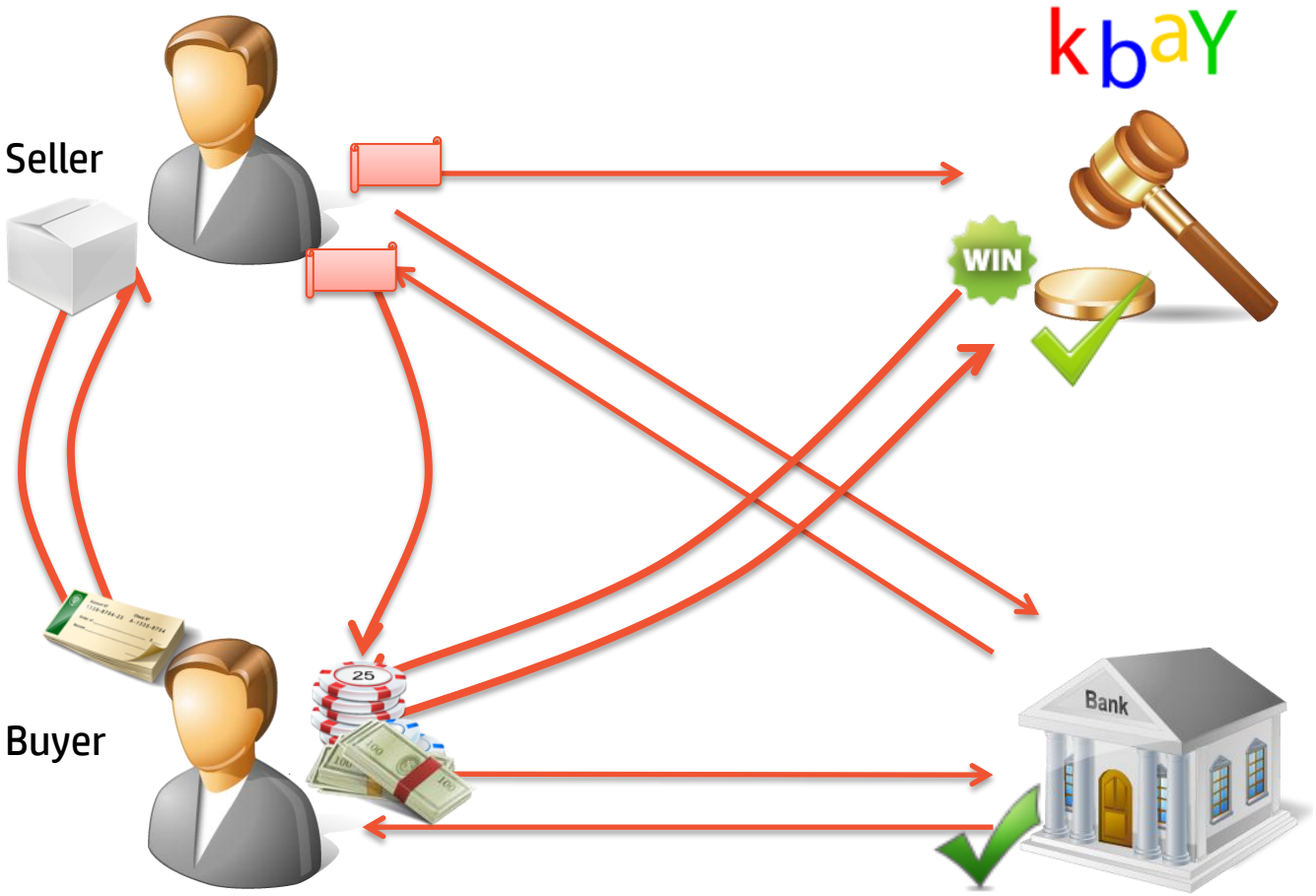
Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Design of Ken



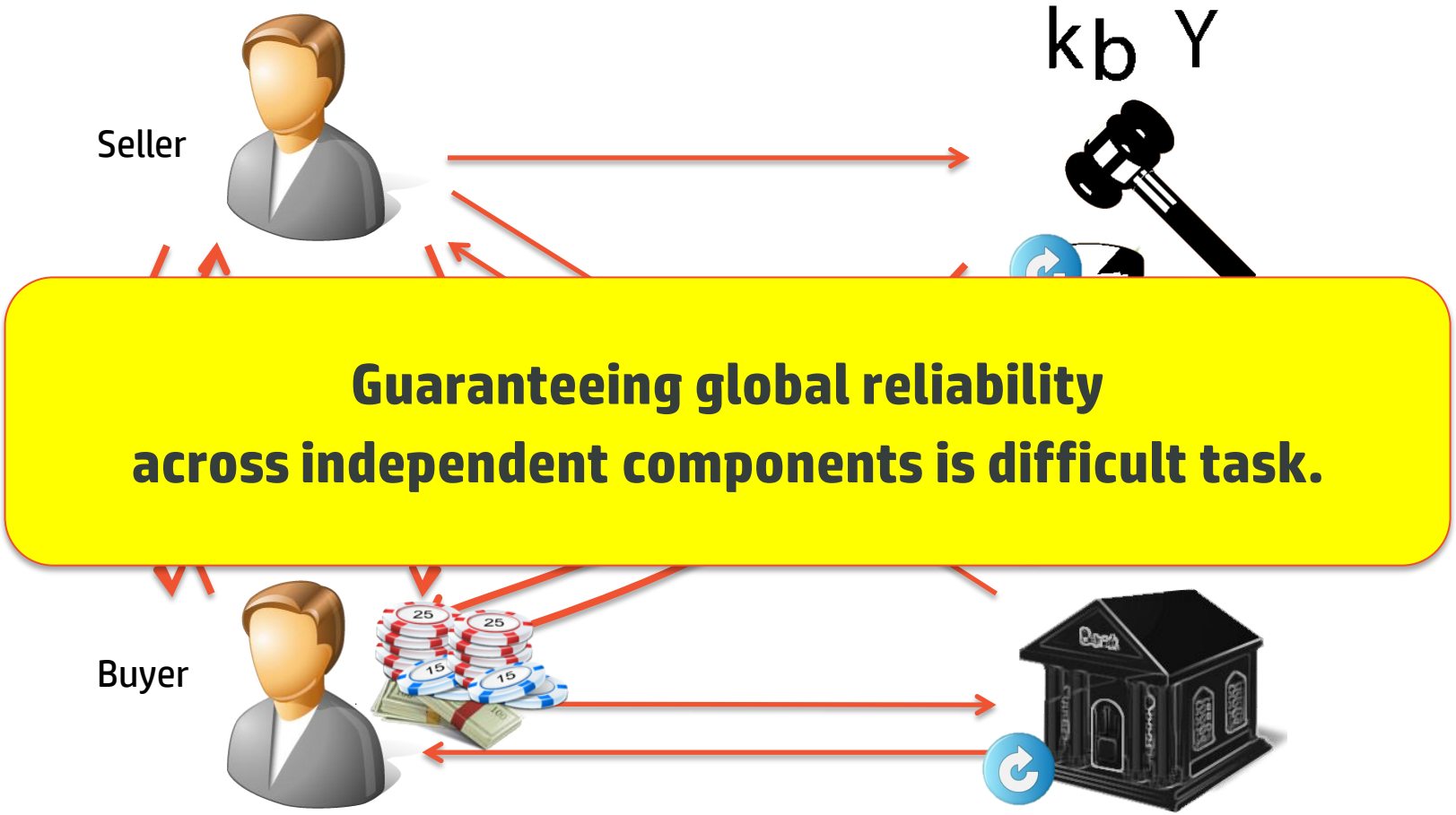
Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Ken: Composable Reliability



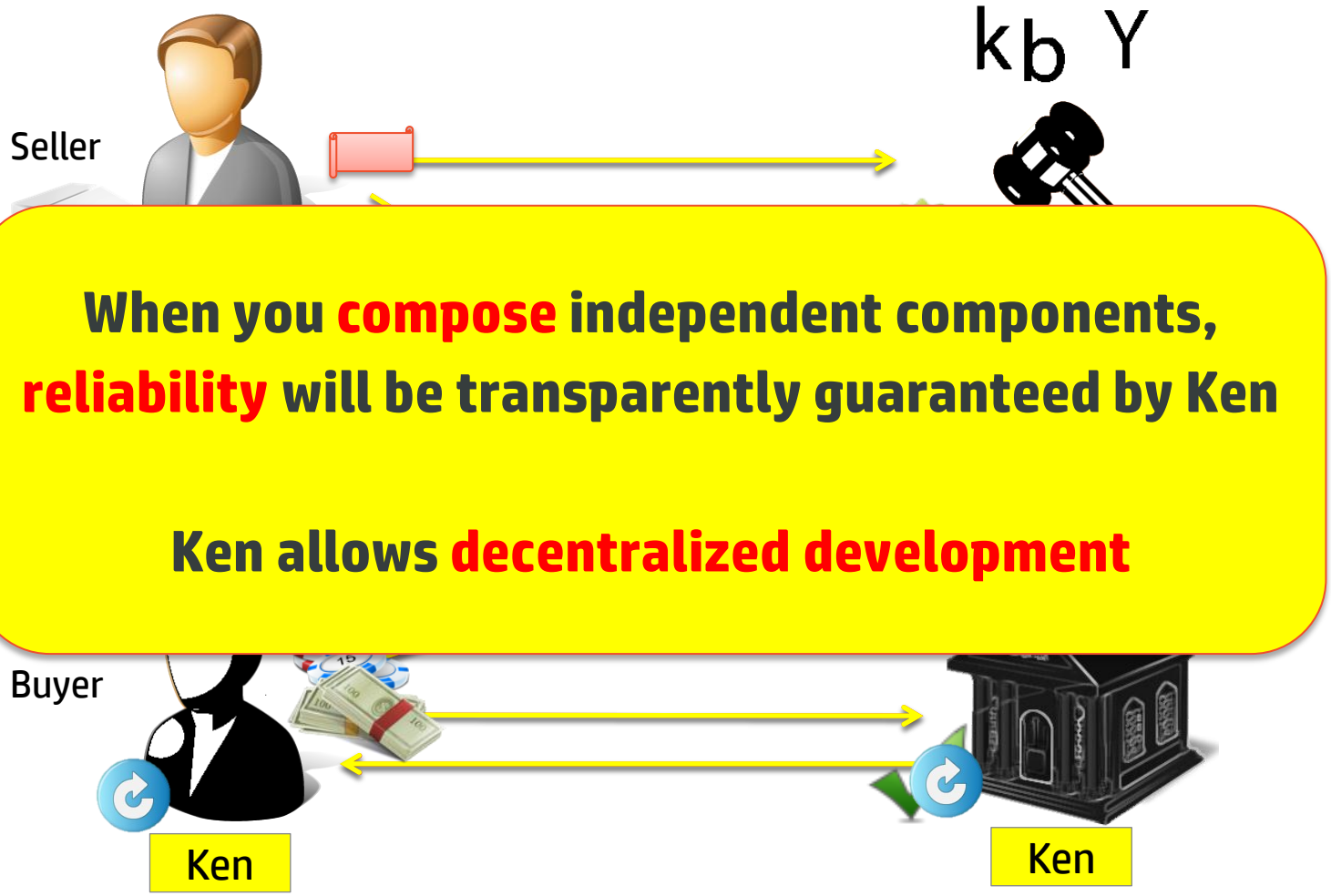
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Ken: Composable Reliability



Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Ken: Composable Reliability



Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Ken Illustrated: "Ping-pong Counter"

```
#include "ken.h"
#include "kenapp.h"

int ken_handler(void *msg, int len,
                kenid_t src) {
  int *cnt;
  if (NULL == msg) {
    cnt = ken_malloc(sizeof *cnt);
    *cnt = 0;
    ken_set_app_data(cnt);
  }
  else {
    cnt = ken_get_app_data();
    *cnt = *(int*)msg + 1;
    ken_send(src, cnt, sizeof *cnt);
  }
  return -1;
}
```

analogue of `main()`

initialization

begin transaction

persistent heap

entry point

incoming message

fire & forget

end transaction

Ken Illustrated: "Ping-pong Counter"

```
#include "ken.h"
#include "kenapp.h"

int ken_handler(void *msg, int len,
                kenid_t src) {
  i
  i
  }
  e
    cnt = ken_get_app_data();
    *cnt = *(int*)msg + 1;
    ken_send(src, cnt, sizeof *cnt);
  }
  return -1;
}
```

analogue of `main()`

Ken programming is simple

1. Implement **ken_handler()** instead of main()
2. Use **ken_malloc / ken_send** instead of malloc / send
3. Use **ken_get_app_data / ken_set_app_data** for entry to persistent heap

fire & forget

end transaction

Mace: "Ping-pong Counter"

Mace with Ken

- Event-driven distributed system language framework
- Used in many projects

```
service PingPong;
services { Transport t; }

state_variables { int cnt = 0; }
messages { pong {int cnt;} }

transitions {
    deliver(src, dest, msg) {
        cnt  = msg.cnt+1;
        route(src, pong(cnt));
    }
}
```

in persistent heap

define state var & messages

incoming message

begin transaction

send message

fire & forget messaging

end transaction

Mace: “Ping-pong Counter”

Mace with Ken

No changes are needed

```
service PingPong;
services { Transport t; }

sta
mes

tra
        route(src, pong(cnt));
    }
}
```

in persistent heap

ages

n

fire & forget messaging

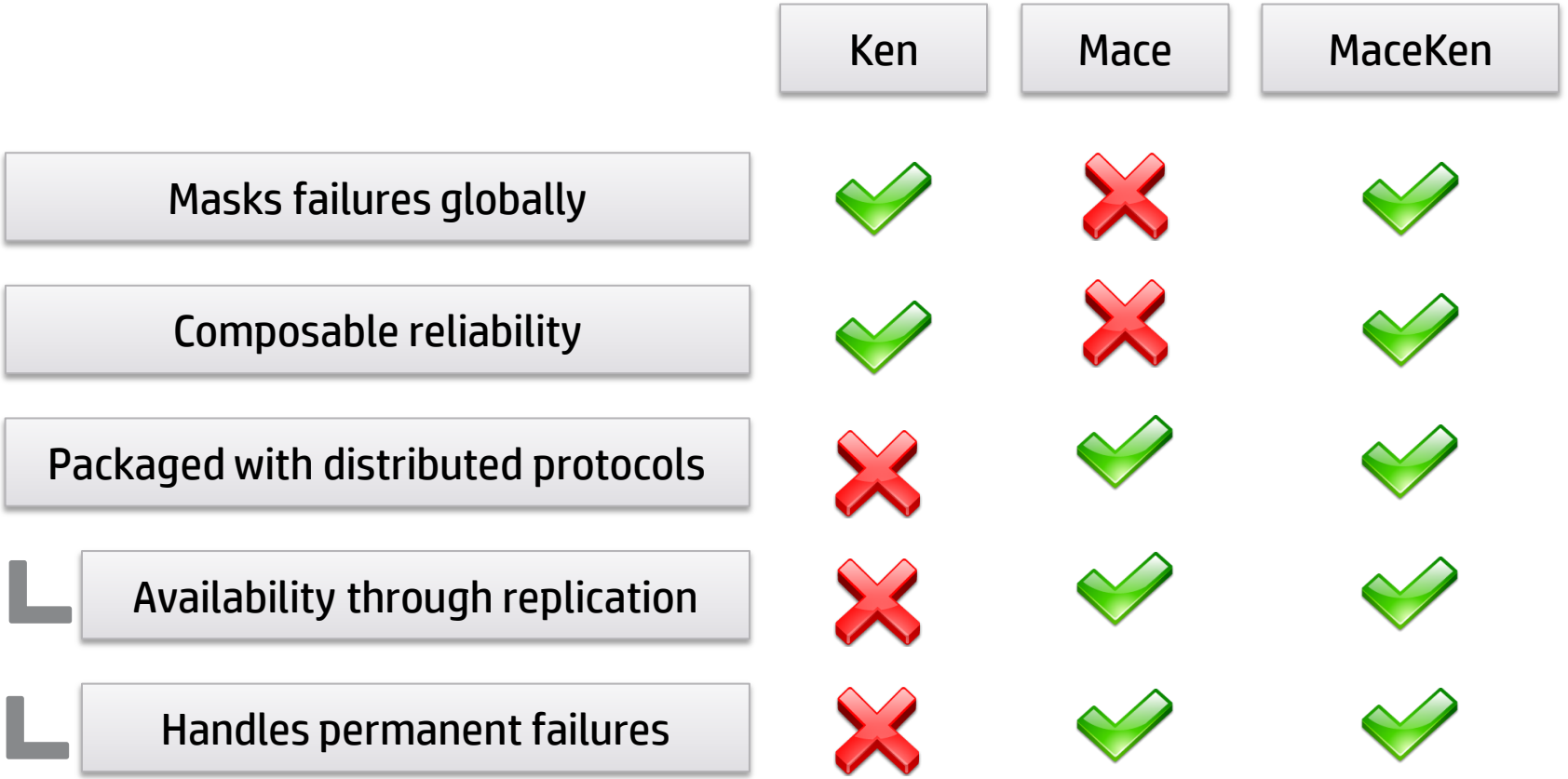
end transaction

You don’t need to change anything for Ken

Reliability and composability comes easily

Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Ken: Integration with Mace



Ken provides new benefits to legacy Mace applications!

Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Implementation

Ken

- C library
- Publicly available

MaceKen

- Modifications to existing Mace runtime libraries
- No changes to existing Mace services

Linux Container (LXC) environment

- Simulating correct power-failure behavior (in paper)

Evaluation

- Microbenchmark
  - Performance test
- kBay
  - Composable reliability (in paper)
- Distributed analysis of 1.1 TB graph
  - Versatility (in paper)
- Bamboo-DHT
  - Failure masking & data survival for legacy Mace app

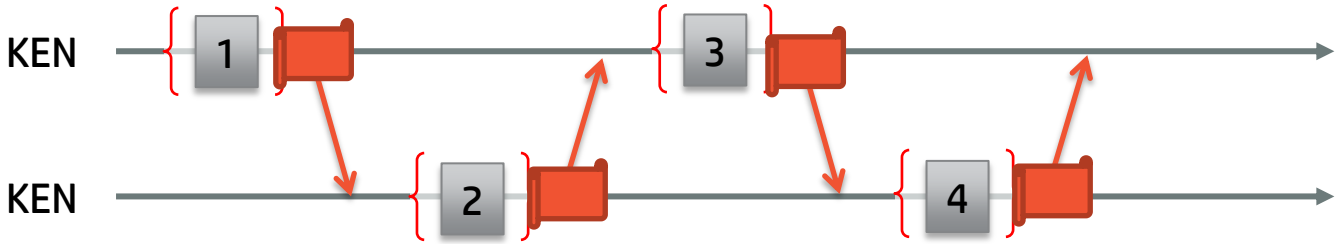
Evaluation: Ken Microbenchmark

Experimental setup

- 16 core, 2.4 GHz Xeon
- 32GB RAM
- Mirrored RAID
  - Two 72 GB 15K RPM disks

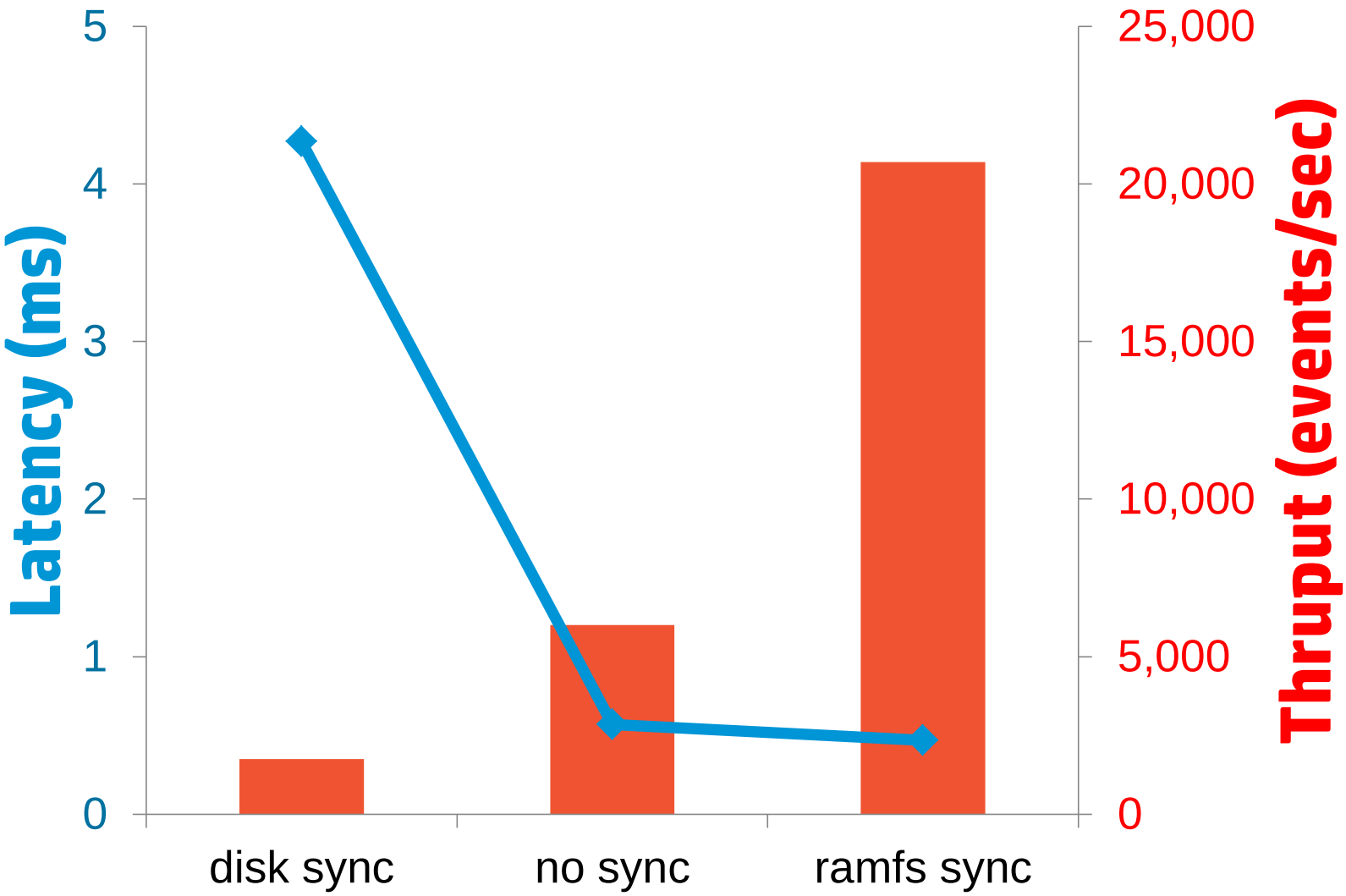
Test

- Ping-pong counter test between two Ken processes



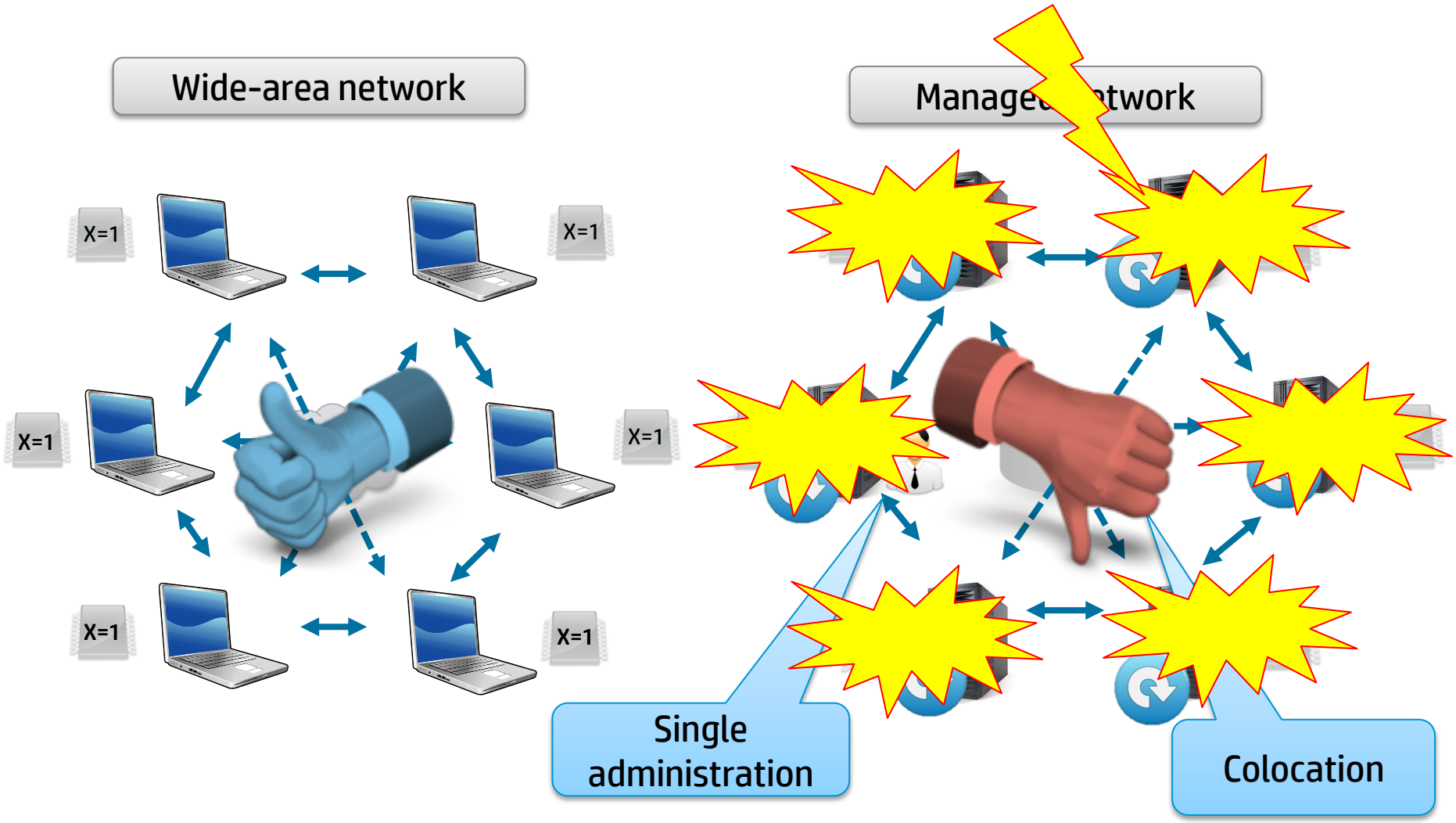
- Measure : latency and throughput

Evaluation: Ken Microbenchmark



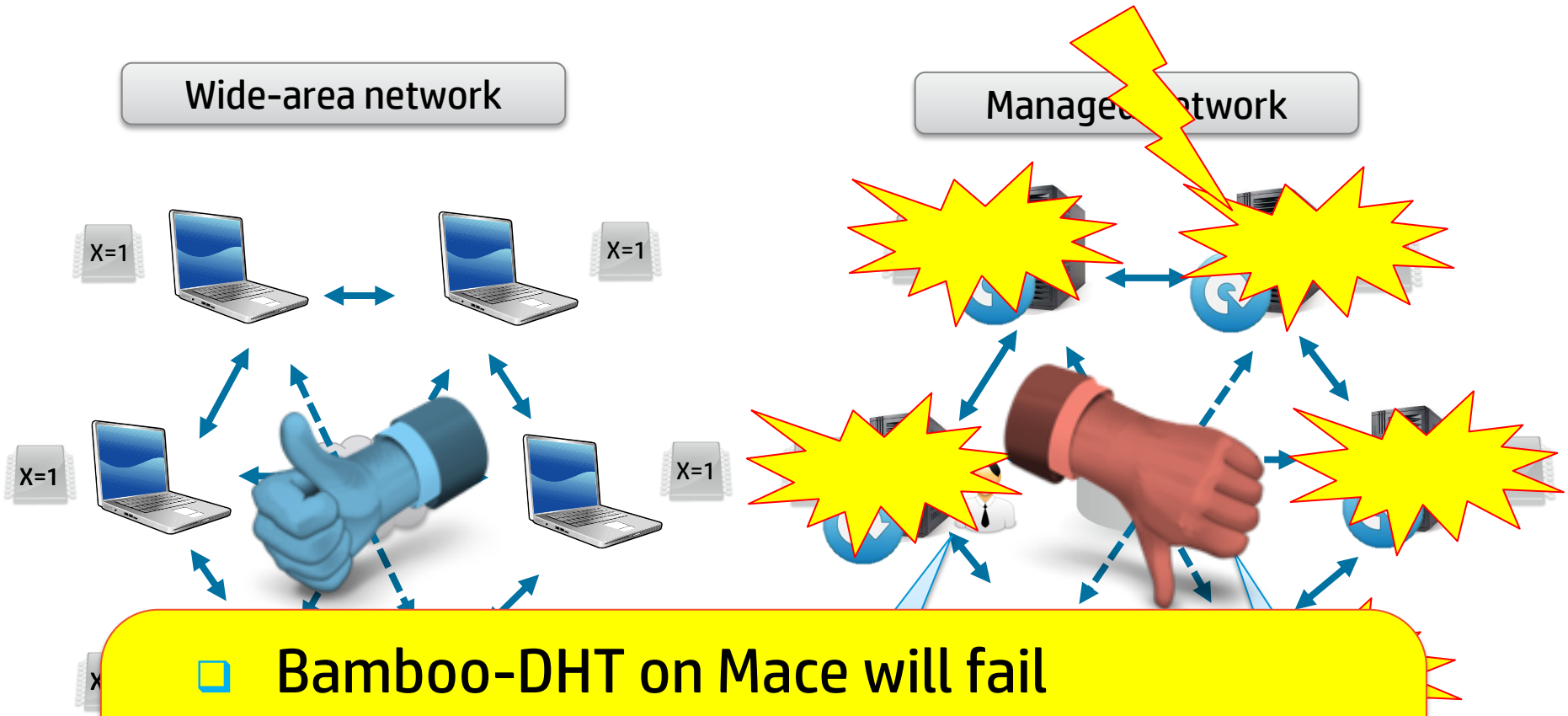
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Evaluation: Bamboo-DHT



Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Evaluation: Bamboo-DHT



- Bamboo-DHT on Mace will fail
- Bamboo-DHT on MaceKen will work
- Mace vs. MaceKen: testing rolling-restart and correlated failure

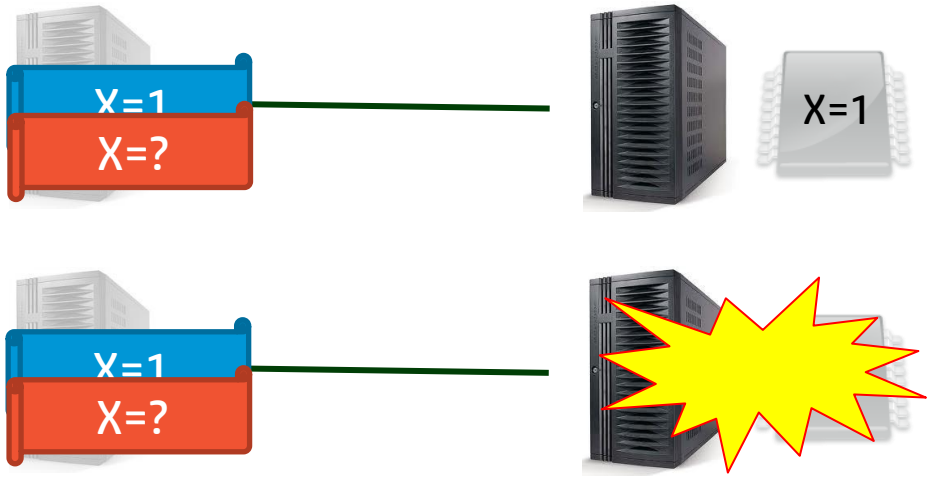
Yoo, Killian, Kelly, Cho and Plite, “Composable Reliability for Asynchronous Systems”, USENIX ATC 2012

Evaluation: Bamboo-DHT

Experimental setup

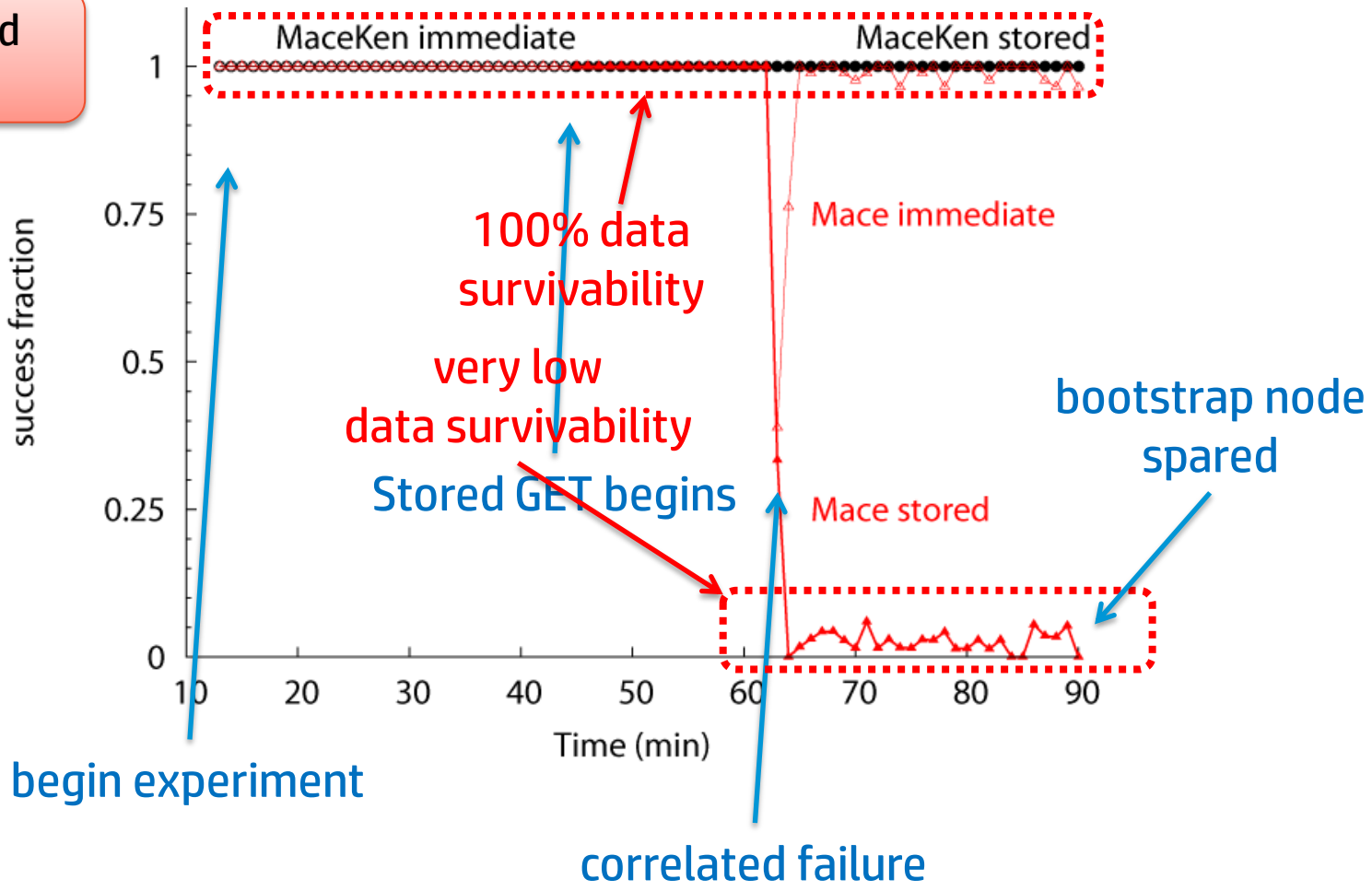
- 12 dual quad-core machines, 8GB RAM
- 4x1Gb Ethernet connections
- 300 DHT nodes on 12 machines

- Measures
  - Immediate-Get
  - Stored-Get



Bamboo-DHT: Crash-restart Failures

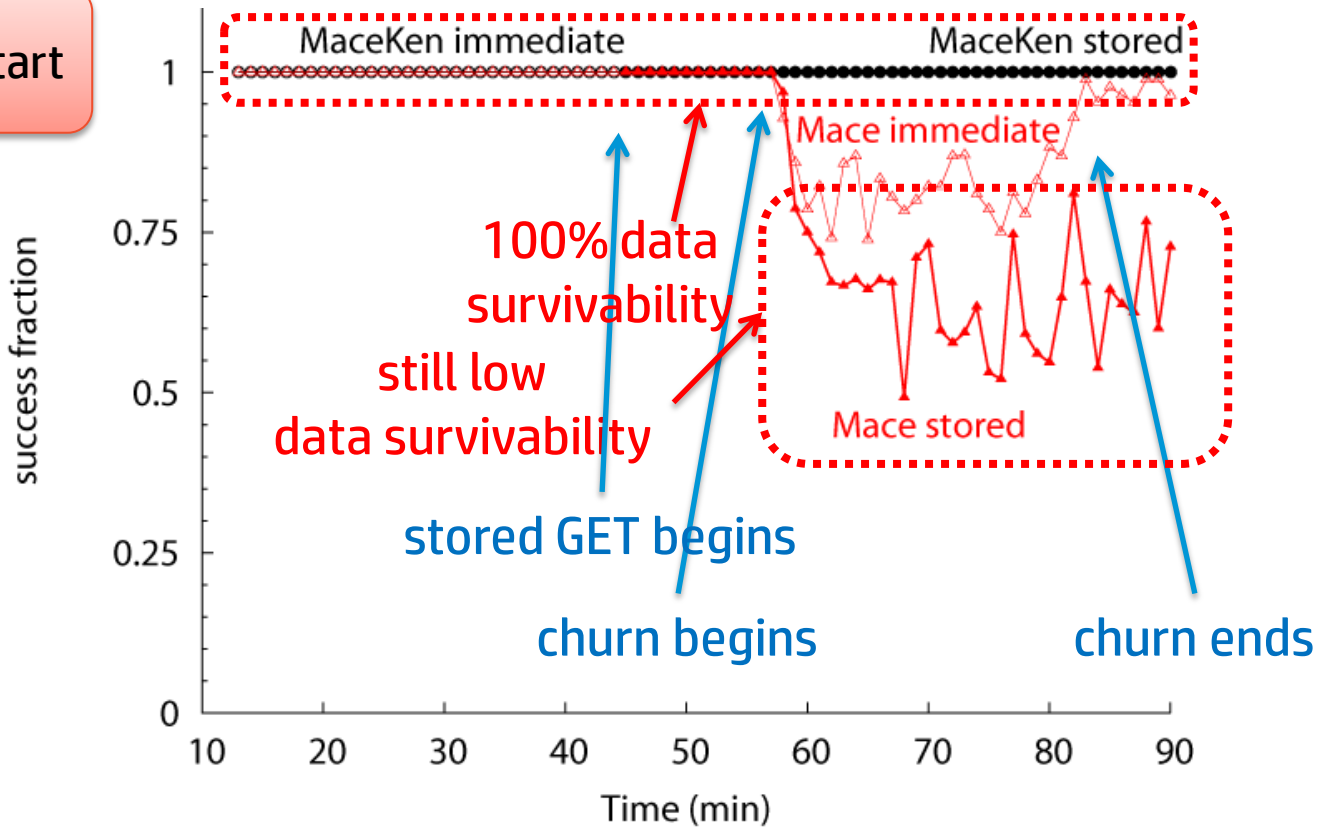
Correlated Failures



Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Bamboo-DHT: Rolling-restart

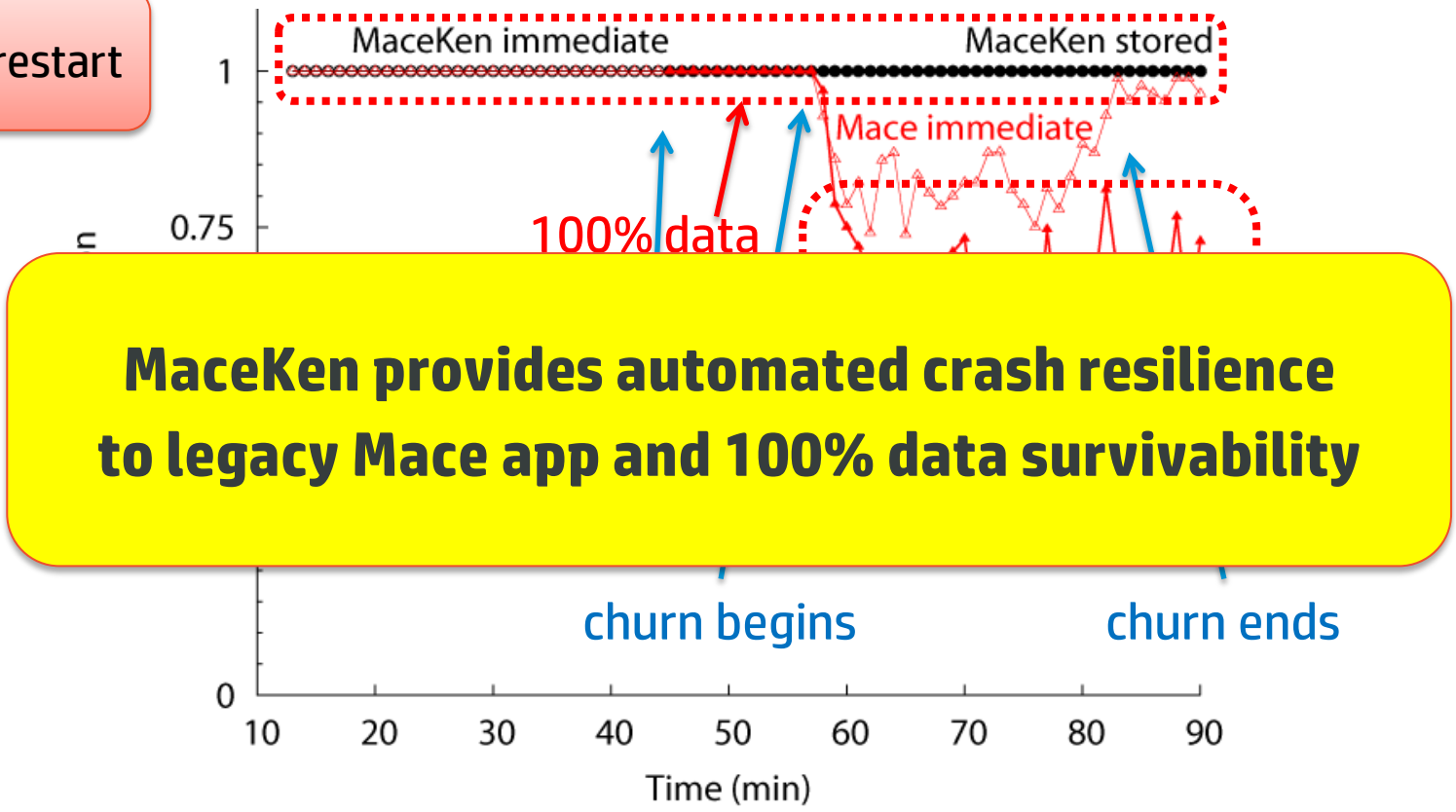
Rolling-restart



Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Bamboo-DHT: Rolling-restart

Rolling-restart



MaceKen provides automated crash resilience to legacy Mace app and 100% data survivability

Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012

Conclusion

Ken provides

1. Reliability
2. Composability
3. Programmability

MaceKen provides

1. Transparency
2. 100% data survivability

Ken available now:

http://ai.eecs.umich.edu/~tpkelly/Ken/

MaceKen available soon:

http://www.macesystems.org/maceken/

Thank you!

Yoo, Killian, Kelly, Cho and Plite, "Composable Reliability for Asynchronous Systems", USENIX ATC 2012