Making an Atomic DHT from Message-passing Reactive KOMPICS Components

Cosmin Arad

cosmin@sics.se

Joint work with **Tallat M. Shafaat** and Prof. **Seif Haridi**

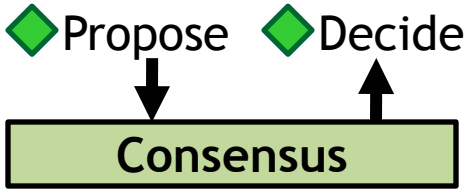
Royal Institute of Technology (KTH), Sweden

Swedish Institute of Computer Science (SICS)

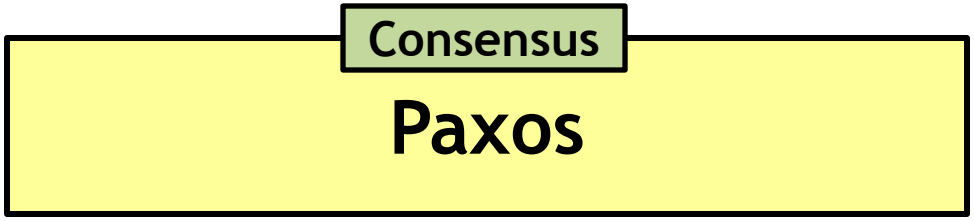
Outline

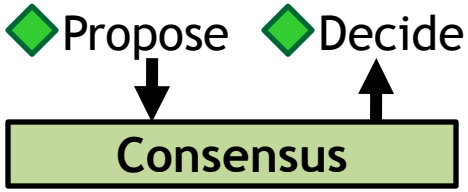
- Part I
  - Kompics: Component model for distributed systems
  - Motivation, model, methodology, tools
- Part II
  - CATS: atomic consistency and partition tolerant DHT
  - Motivation, solution, architecture, early evaluation
- Current and future work

Motivation for KOMPICS

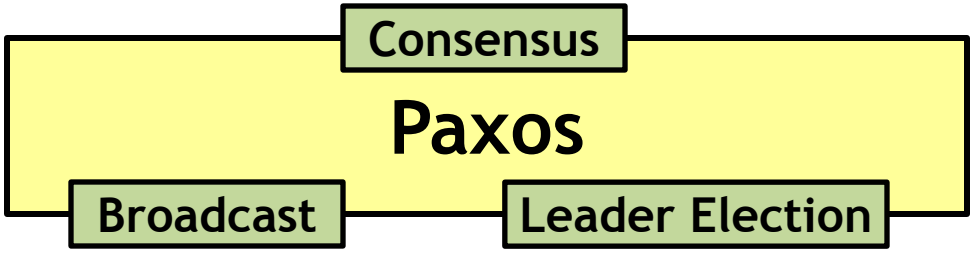


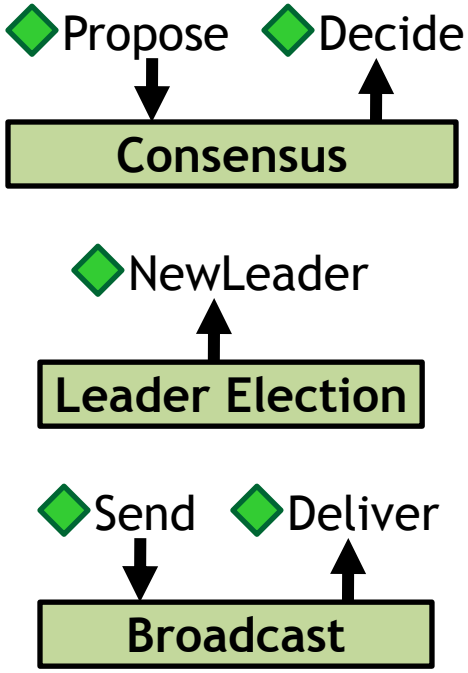
Motivation for KOMPICS



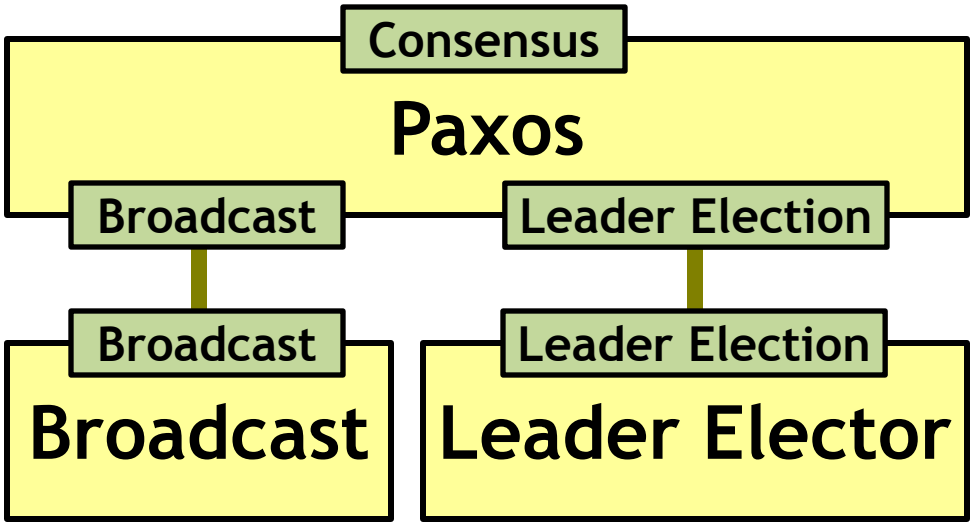


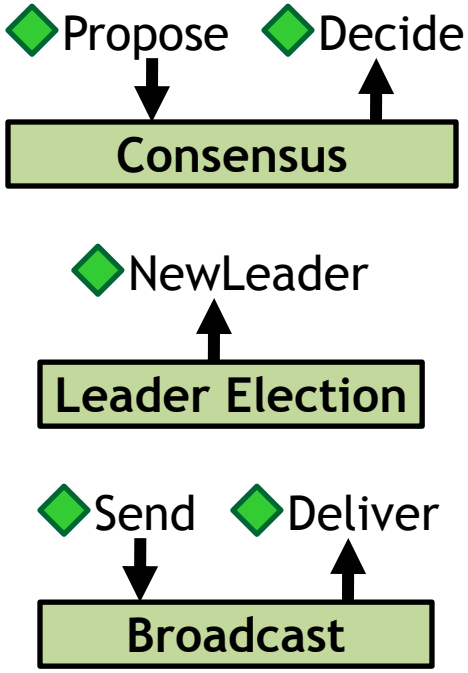
Motivation for KOMPICS



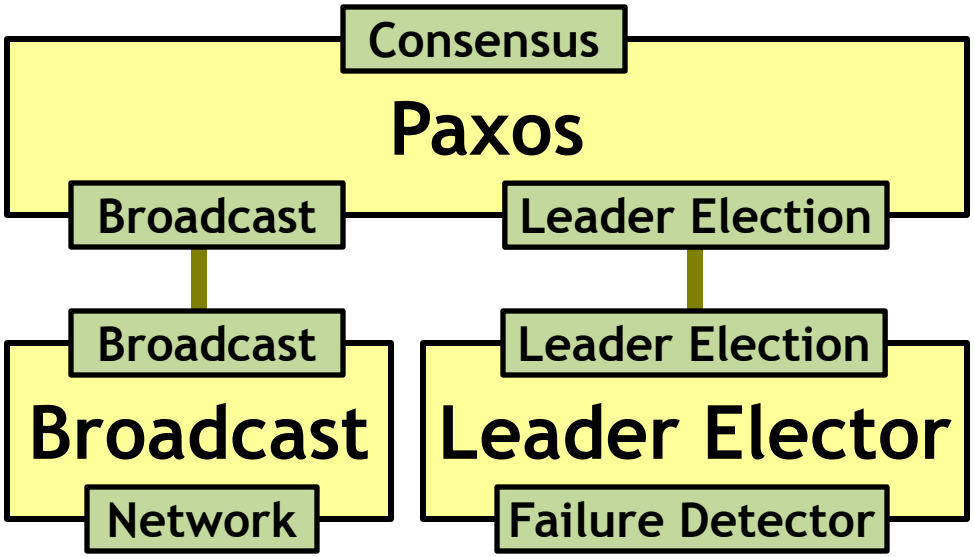


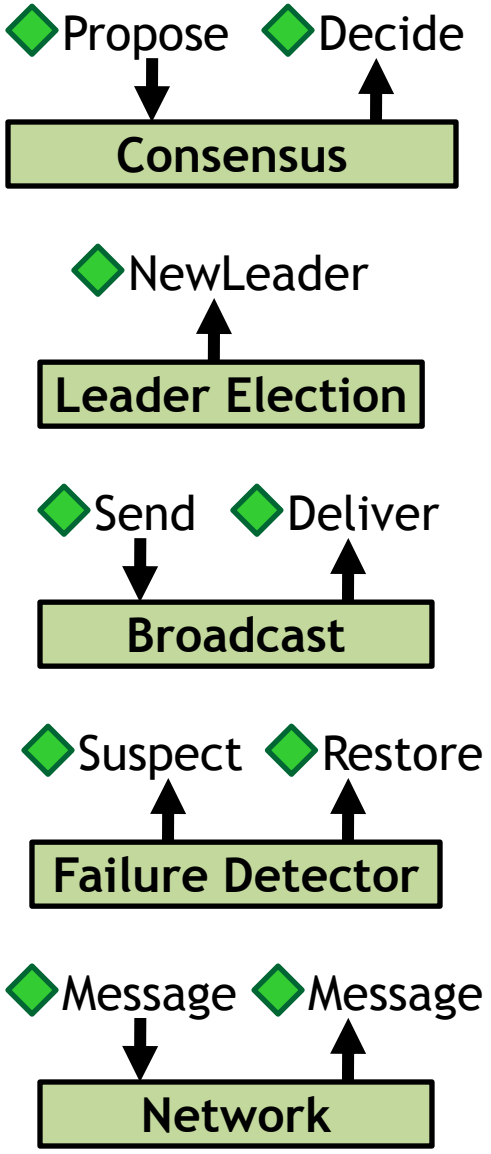
Motivation for KOMPICS



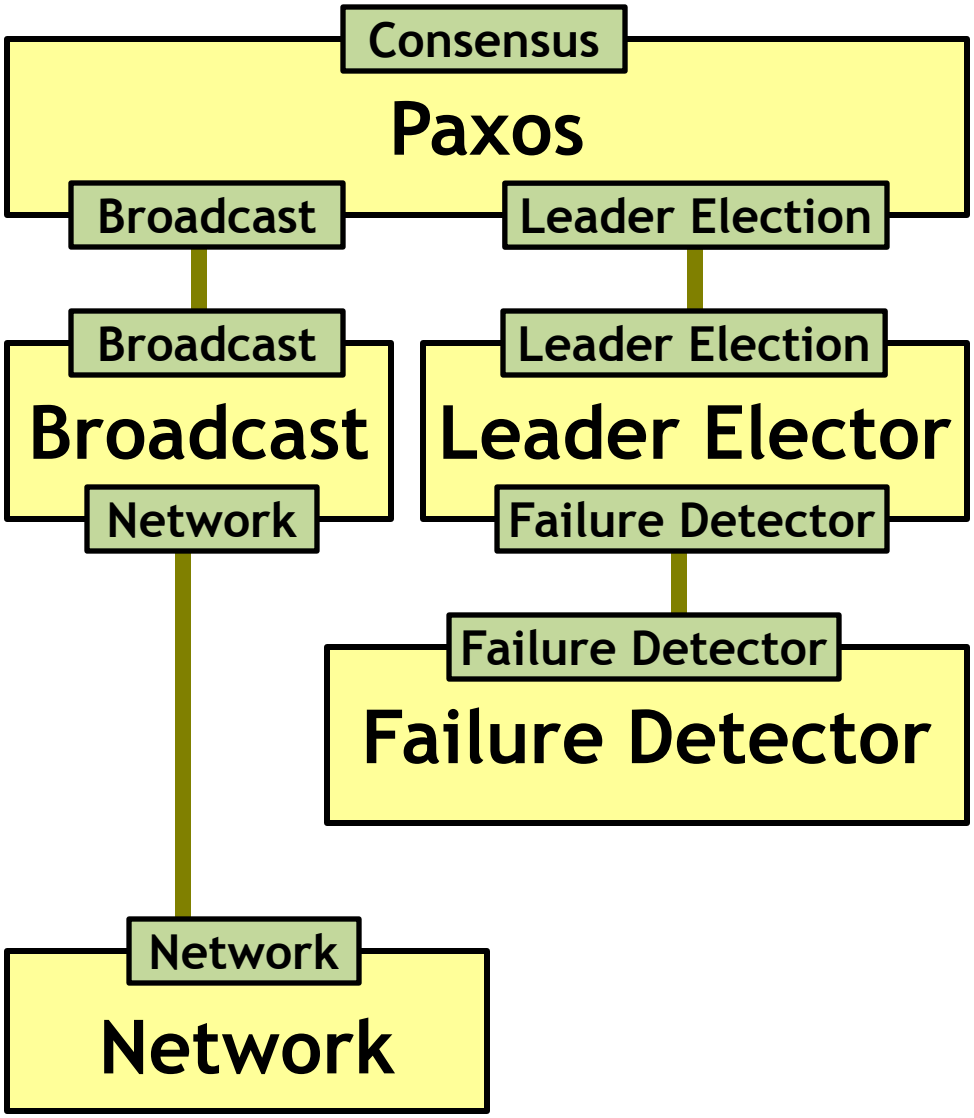


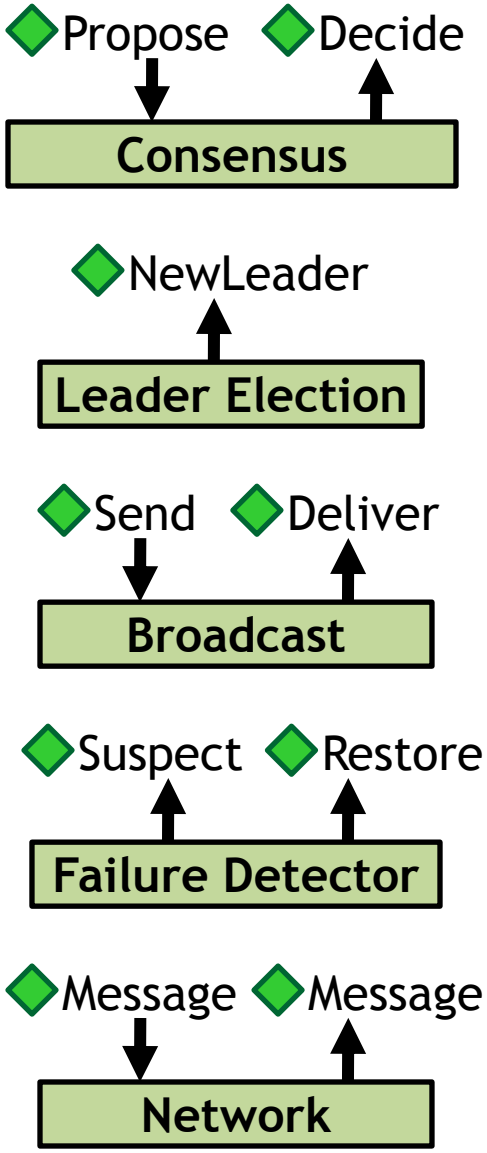
Motivation for KOMPICS



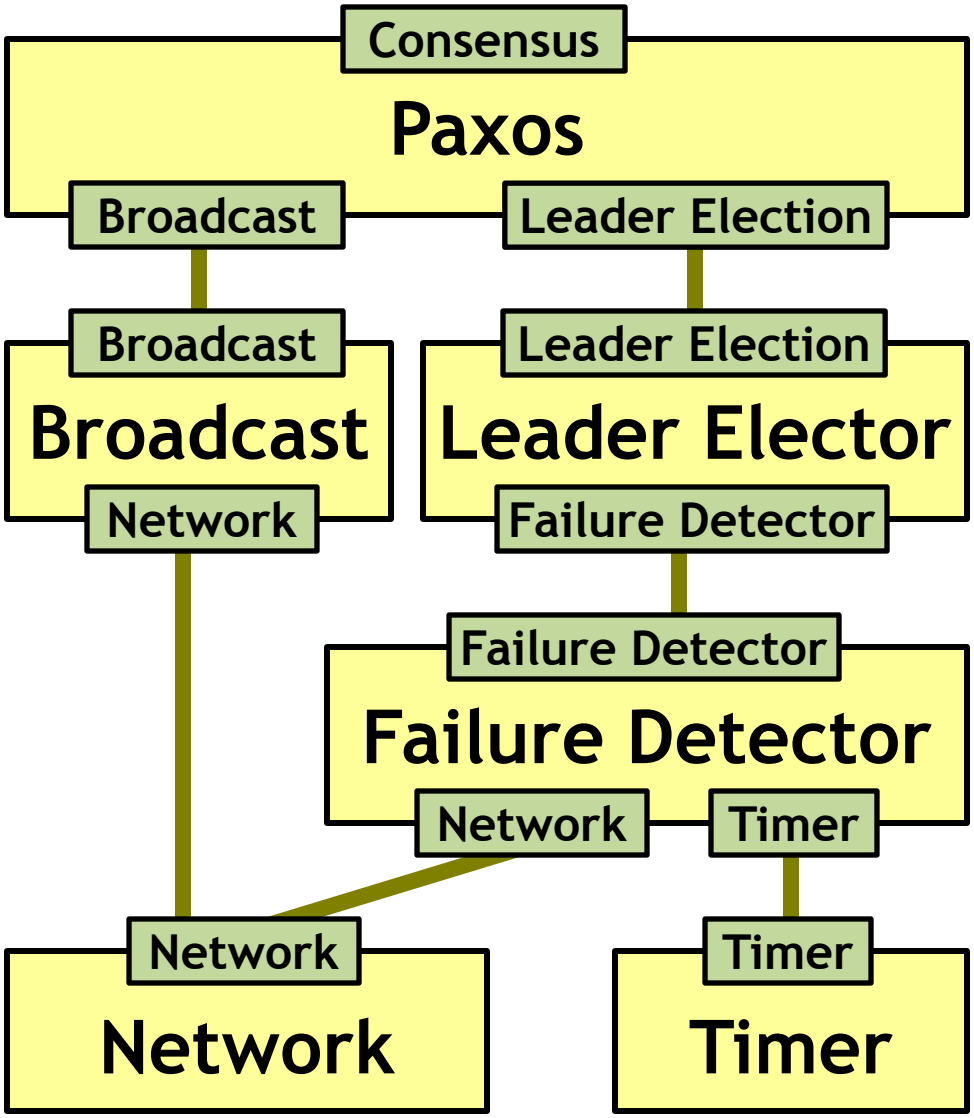


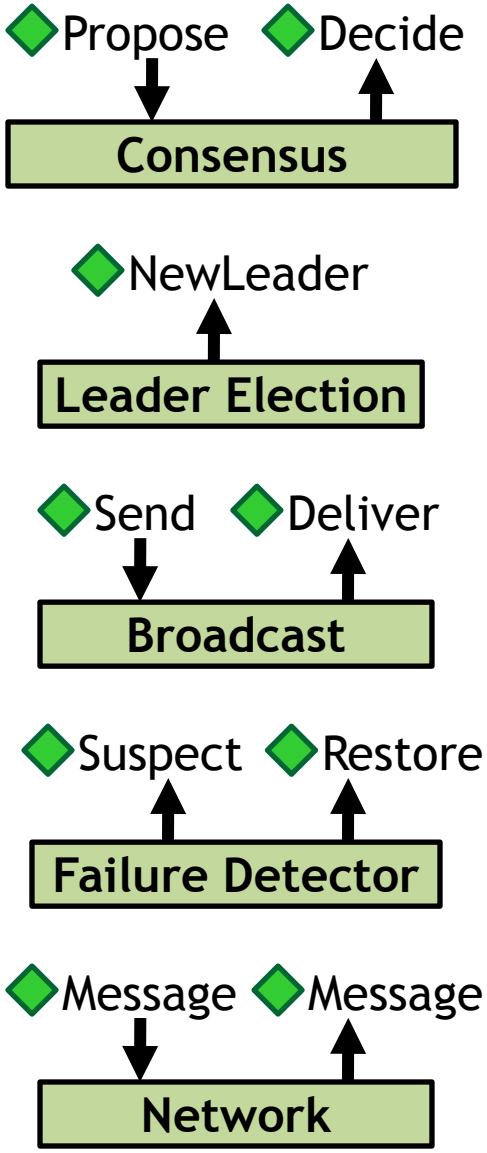
Motivation for KOMPICS





Motivation for KOMPICS





KOMPICS characteristics

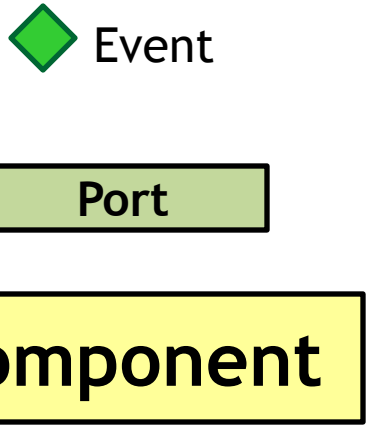
- Hierarchical composition
- Message-passing
  - compositional concurrency
- Publish-subscribe interaction
  - loose coupling
- Dynamic architecture reconfiguration
- Event-driven programming model

Related models, inspiration

- Component models: Fractal, OpenCom, ArchJava, ComponentJ
  - Synchronous interface calls vs. message passing
- Protocol composition frameworks: x-Kernel, Ensemble, Isis, Horus, …, Bast, Live Objects
  - (static) layered vs. (dynamic) hierarchical composition
- Actor models: Erlang, Kilim, Scala, Unix pipes
  - Flat vs. hierarchical architecture
- Process calculi: π-calculus, CCS, CSP, Oz/K
  - Synchronous vs. asynchronous message passing

KOMPICS concepts

- Event
- Port
- Component
- Channel
- Handler
- Subscription
- Publication / Event trigger

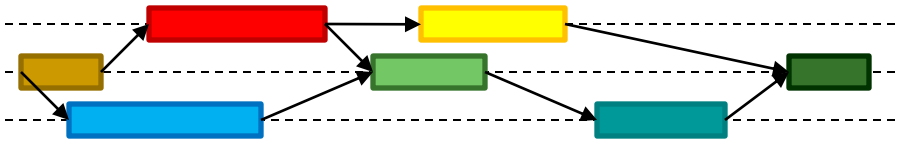


Execution profiles

- Production deployment
  - Multi-core component scheduler (work stealing)
  - One distributed system node per OS process
- Local execution
  - Entire system in one OS process
  - Interactive testing
- Local simulation
  - Testing and stepped debugging
  - Reproducible simulation (of entire system)

Experiment scenarios DSL

- Define parameterized scenario events
  - Node failures, joins, system requests, operations
- Define *"stochastic processes"*
  - Finite sequence of scenario events
  - Specify distribution of event inter-arrival times
  - Specify type and number of events in sequence
  - Specify distribution of each event parameter value
- Scenario: composition of "stochastic processes"
  - Sequential, parallel:



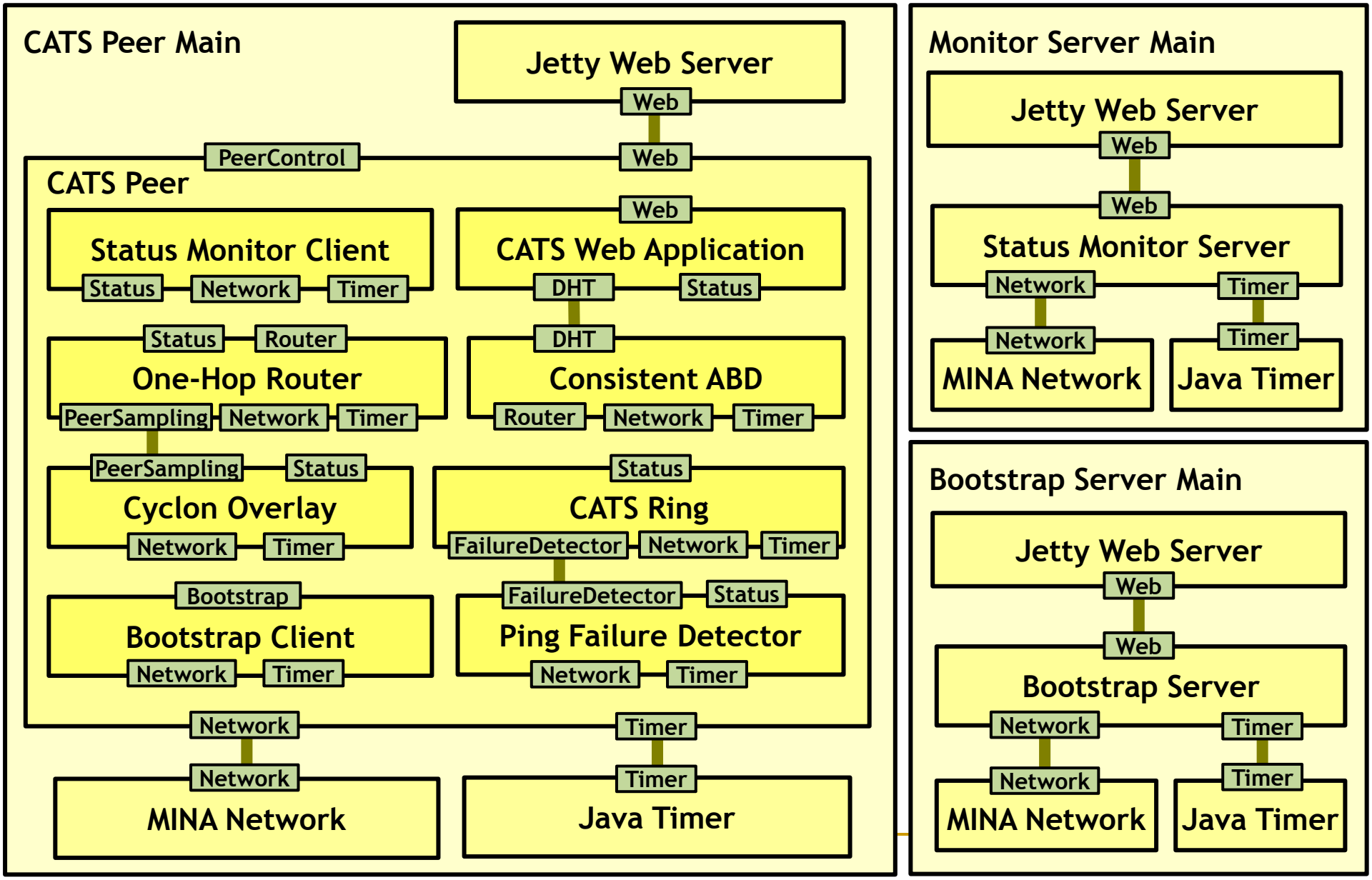
Motivation for CATS

- Scalable meta-data storage for cloud systems
  - provide *Consistency* and *Partition tolerance* (CP)
- Consistent hashing
  - Incremental scalability, simplicity, self-organization
- Reconfigurable atomic shared-memory registers
- Can we combine them, have the best of both?
  - Yes, but not trivially…
    - Problem with false failure suspicions and network partitions for consistent hashing and quorum-based algorithms
    - Introduce *consistent quorums* as a solution

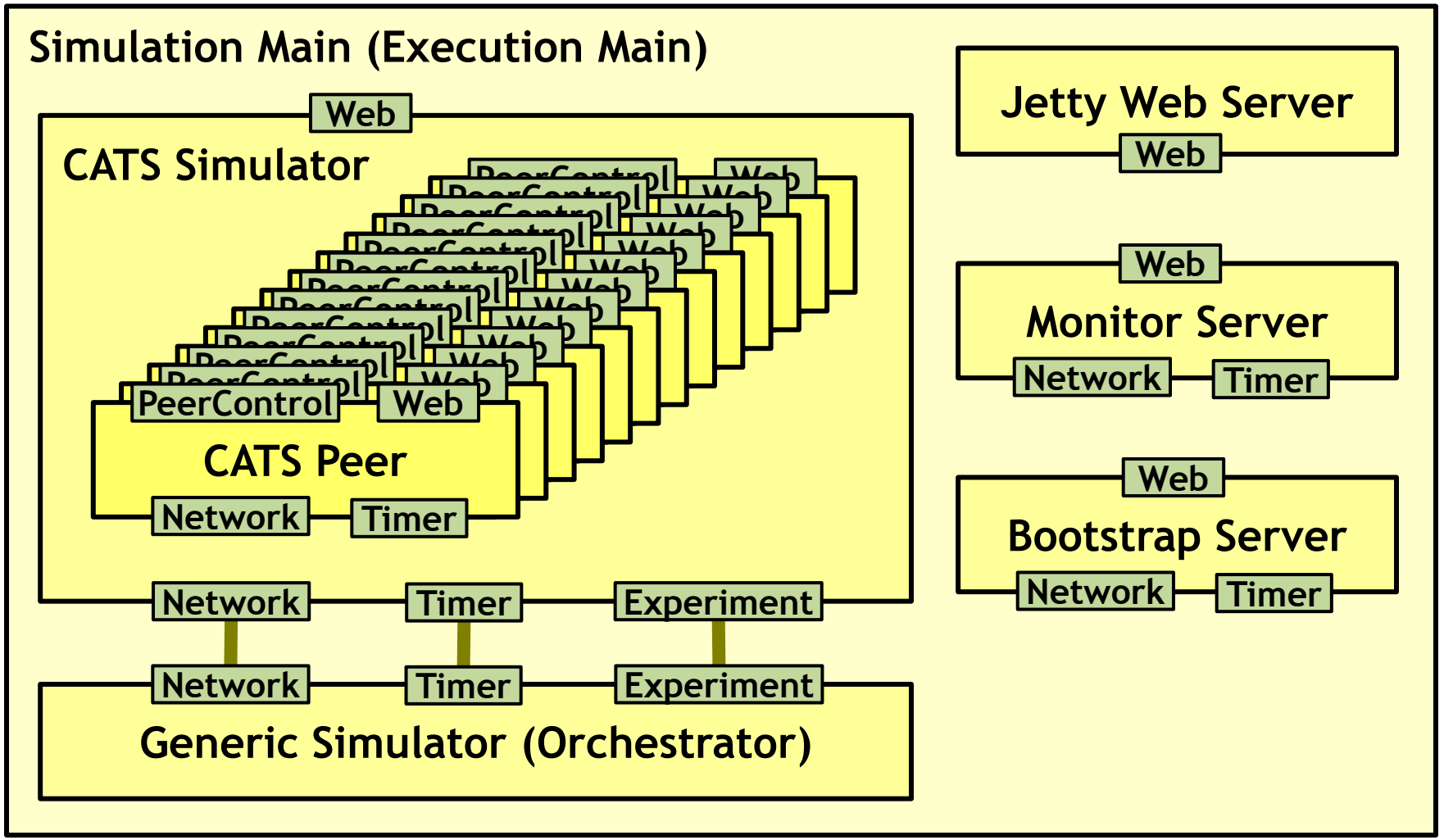
CATS solution

- Consistent hashing ring (e.g., Chord'01)
  - Successor-list replication
  - One-Hop lookup routing
  - Cyclon peer sampling
- Shared memory atomic register (e.g., ABD'95)
  - Using *consistent quorums*
- Paxos-based reconfiguration (e.g., RDS'09)
  - Using *consistent quorums*

Production deployment architecture



Simulation architecture



Early CATS evaluation

- 7 server machines, 16 client machines

Servers	Clients	PUTs/client	Throughput [ops/sec]	Latency [ms]
7	7	500	1600	3
7	8	500	1750	5
7	9	500	1900	9
7	10	500	2150	100
14	10	500	2500	10

- Flash crowd: 21 server nodes, 14 client nodes

PUTs/client	Throughput [ops/sec]	Latency [ms]
500	3100	15
1000	3800	35
2000	4400	100
4000	4600	1200

Current and future work on CATS

- Comprehensive performance evaluation
  - Latency vs. throughput
  - Scalability
  - Elasticity
  - Using well-known benchmarks (e.g. YCSB)
- Formal correctness proof for core algorithms
- Transactional DHT using Paxos Commit
- Scalable Replicated State Machines

KOMPICS Summary

- Message-passing, hierarchical component model
- Good for distributed abstractions & algorithms
- **Multi-core** hardware exploited for free
- Hot upgrades through dynamic reconfiguration
- **Same code** used in *simulation*, *local execution* and *production deployment*
- DSL to specify complex experiment scenarios
  - gossip-based and structured overlays, NAT traversal
  - P2P media streaming, video-on demand

Future work on KOMPICS

- Priority scheduling for components and events
- C porting for kernel modules as components
  - Leverage inter-core message passing in Barrelfish
- Patterns for dynamic reconfiguration
  - Transactional component reconfiguration
- Mechanisms for deployment, dependency mgt.
- Scala implementation for reduced verbosity

THANK YOU!

Q&A

KOMPICS docs: http://kompics.sics.se/

CATS live demo: http://cloud6.sics.se/