



FTSS: A Fault-Tolerant Storage Substrate for R3 Routers

Andrei Agapi

ASFR09, Amsterdam, 10/26/2009

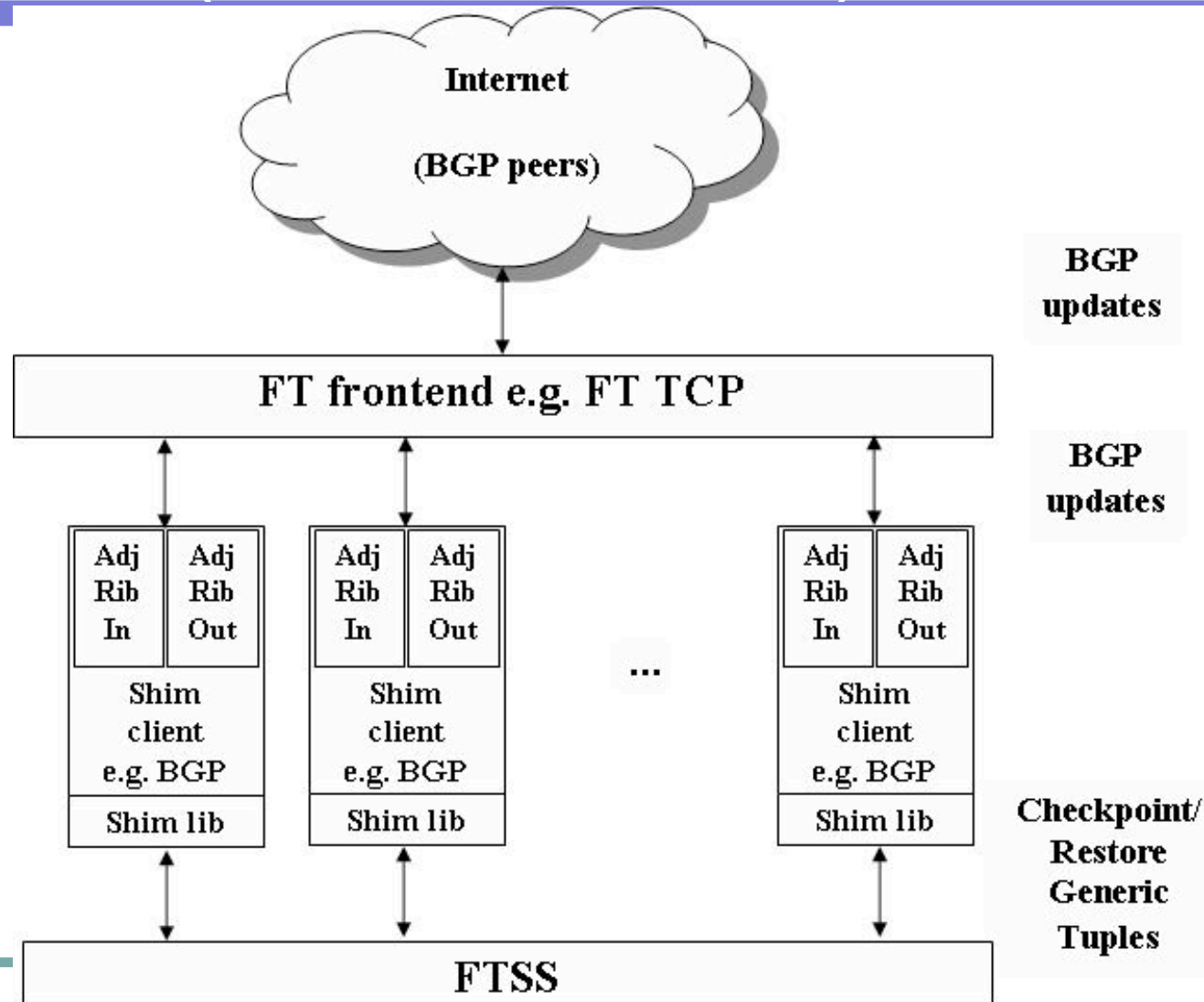
Context

- Today's PoPs and NG routers look very much like well-connected datacenters/clusters
 - 10s -> 100s of line cards, millions of interfaces
- Very well connected (GE, CRS interconnect)
- Lots of resources to harvest/integrate, from both scaling and FT perspectives
 - E.g.: RAM
- Scale up capacity, while increasing FT / robustness and keep good latencies
- Paradigm shift to “distributed, reliable routers” (R3)

FTSS Quick Fact Sheet

- We're trying to build a storage substrate for NG routers that is:
 - Shared among router nodes and can be scaled beyond (e.g. inter-CRS?)
 - FT / Malleable to random failures
 - Fast!
 - Various consistency semantics
 - Fit target platform well: e.g. line cards, RAM constraints, cheap multicast etc

Application: The Shim (BGP illustrated)



Application: The Shim [2]

- Routing app instances (e.g. BGP or IS-IS speakers) run on various RPs a/o line cards
 - These are shim apps (or 'clients')
 - Replicated or not really (e.g. multi version)
- All work on input from peers (e.g. from FT TCP)
 - E.g. BGP updates to/from Adj-Rib-Ins/Outs
- State stored into FTSS (checkpoint)
 - E.g. consistently replicated logs as in the replicated state machine model
- Shim clients can transparently fail/be restarted from FTSS logs + input with app flow (e.g. BGP) unaffected as per outside perspective

Why DHTs ?

- Scalability: nice but (for now) not as critical as for e.g. p2p
- FT: big issue
 - Assume N-modular fault aware stance (w.r.t. fail-stop thus far)
 - Symmetric replication, malleability to random fail-stop failures
 - Encompass: failure detection, replication, load balancing, discovery & self (re)configuration
 - Dynamic resource harvesting / management

Why DHTs [2] ?

- A good extendable framework to further support:
 - Various consistency mechanisms
 - Durability/Availability/Performance tradeoff
 - May extend to other failure models, e.g. Byzantine voting etc
 - Enable components to assume a “mutually suspicious” stance

DHT properties

- They provide good symmetric resilience, tunable FT properties
 - at least on fail-stop assumptions
- They tend to be 'PnP', or at least easy to configure
- Potentially scale very well beyond
- Load balanced: can harvest resources well; very useful to harvest RAM
- Malleable, self configure at failures

“Why not DHTs?” - limitations

- Traditionally not built for latency
 - Used before in datacenter-like setups (e.g. Dynamo)
 - generally: TPT+FT; ours: LAT+FT
- Traditionally read-intensive
 - Ours: mostly write intensive
- Traditionally: eventually consistent
 - Ours: provides serializability
 - ~ malleable DSM

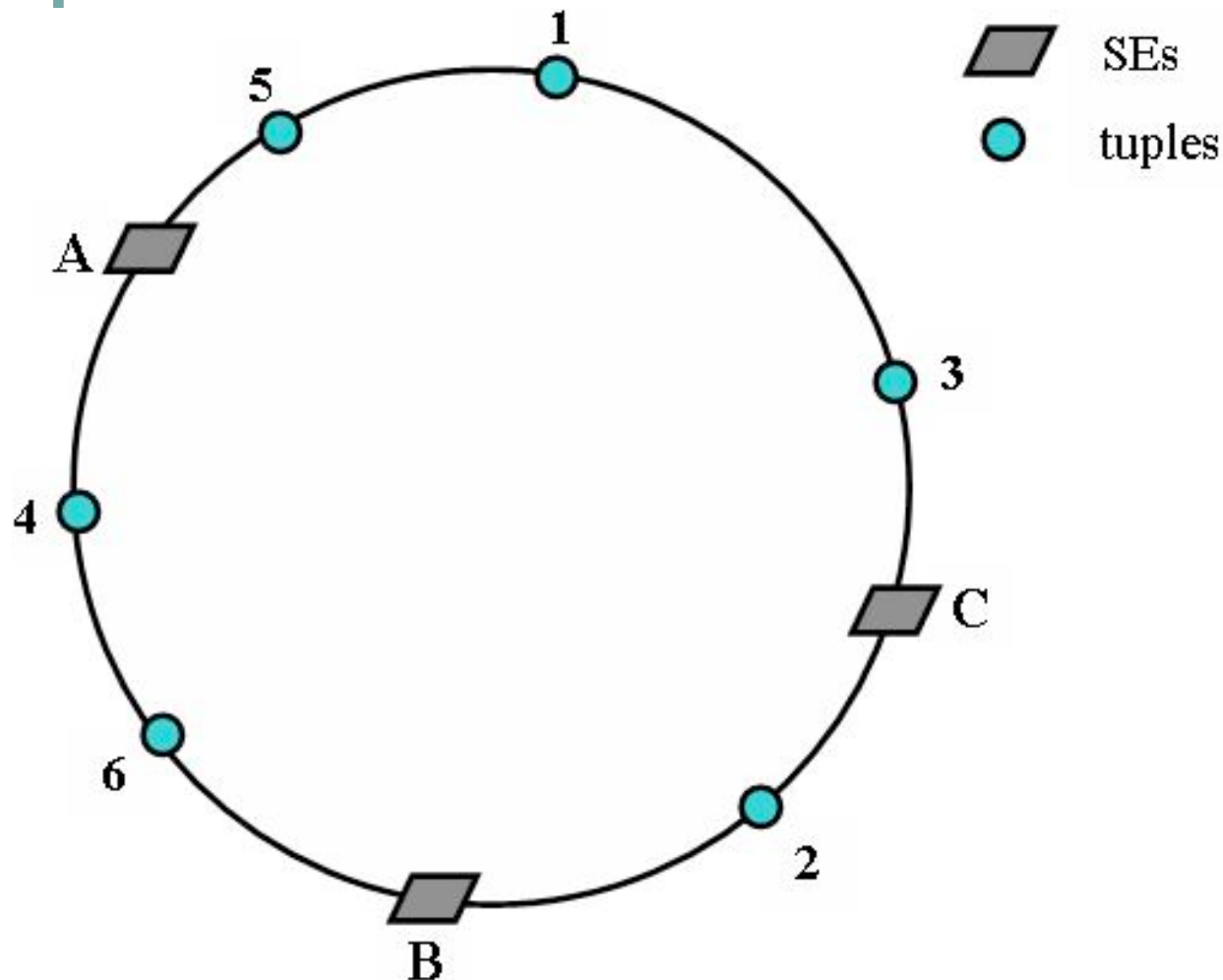
“Why not DHTs?” - limitations [2]

- Traditionally designed for unicast media (WANs)
 - Ours: should leverage intra-CRS multicast
- Traditionally problem with range/complex queries:
 - We may need such for some scenarios



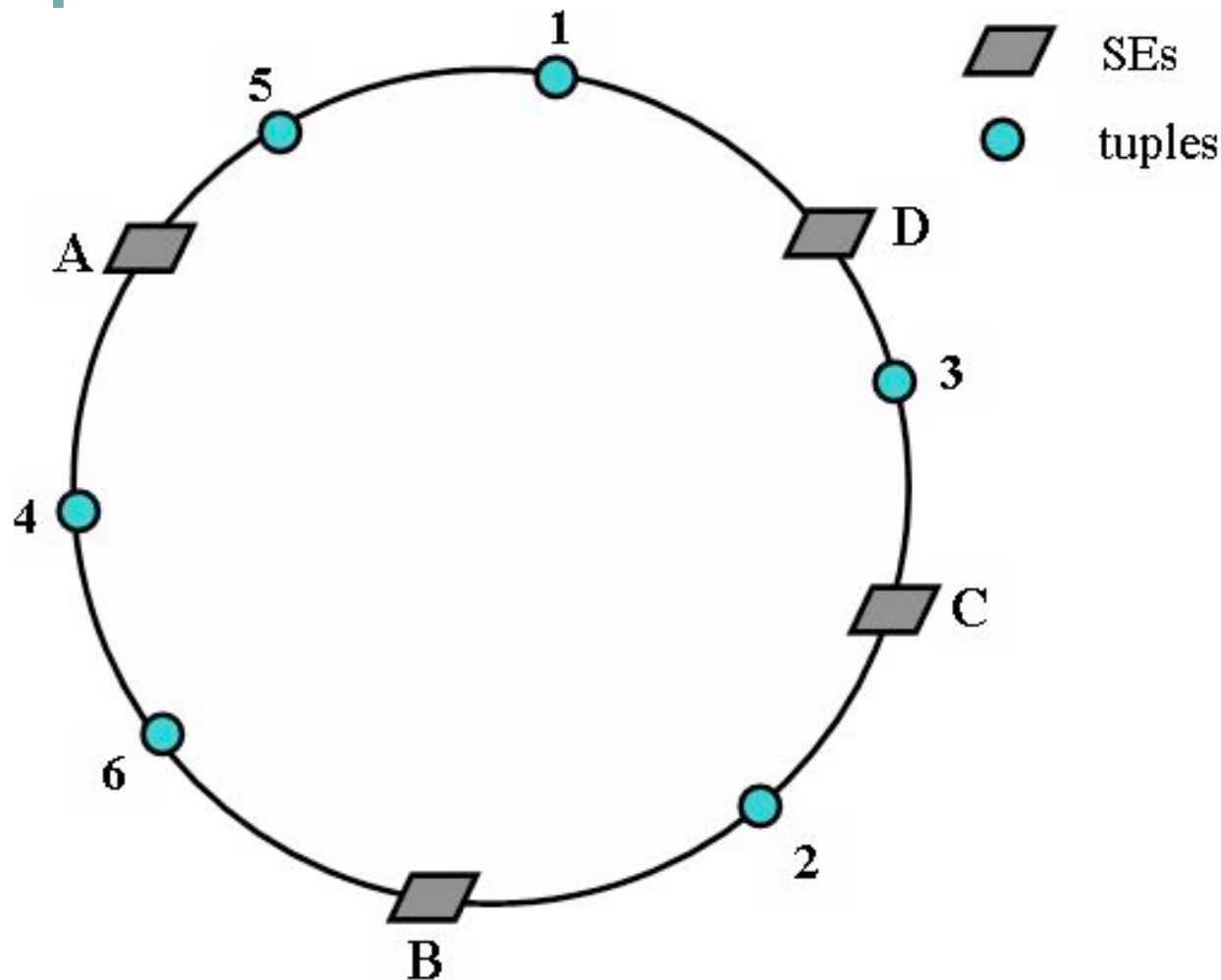
Short baseline story on Chord

Consistent Hashing



- Tuples, SEs hashed on same circle
- Each SE stores tuples immediately preceding it:
A \leftarrow 6,4
C \leftarrow 5,1,3
B \leftarrow 2

Consistent Hashing [2]



- As SEs join/fail min amount of tuples are remapped:

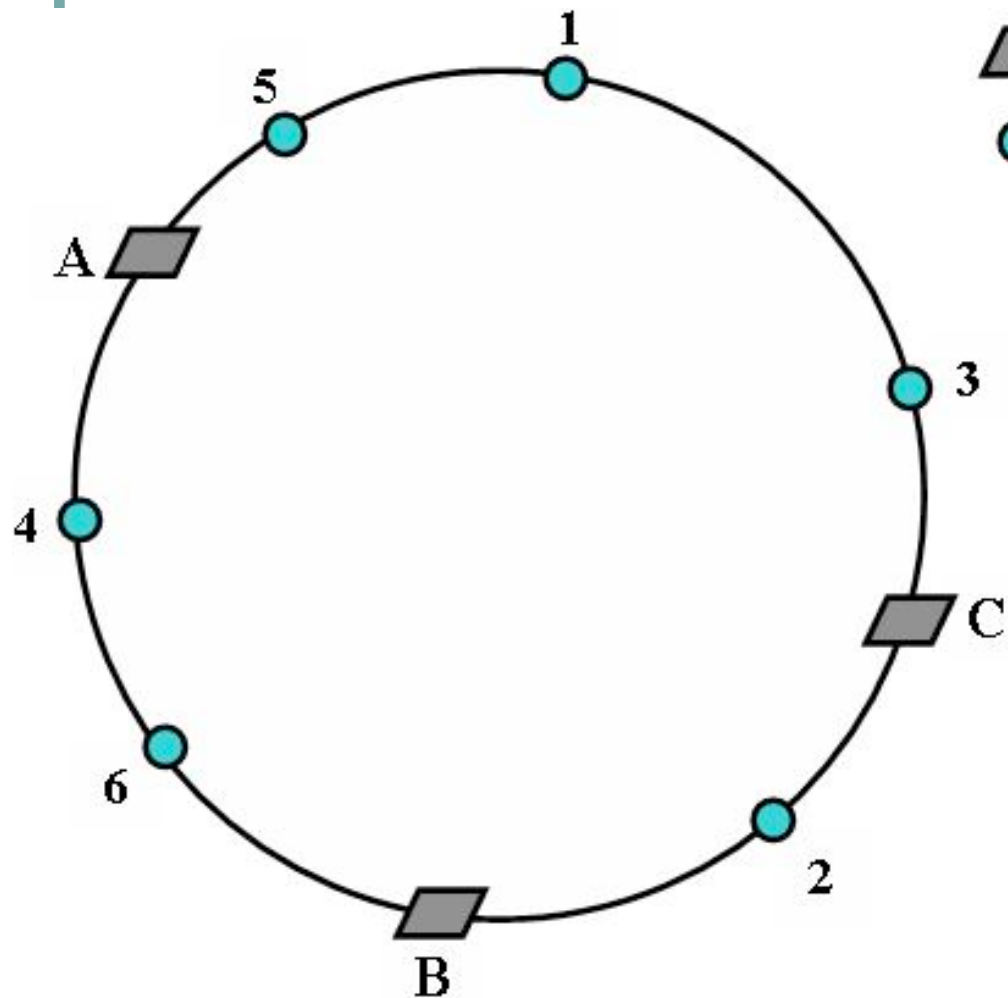
A ← 6, 4

C ← 3

B ← 2

D ← 5, 1

Consistent Hashing [3]

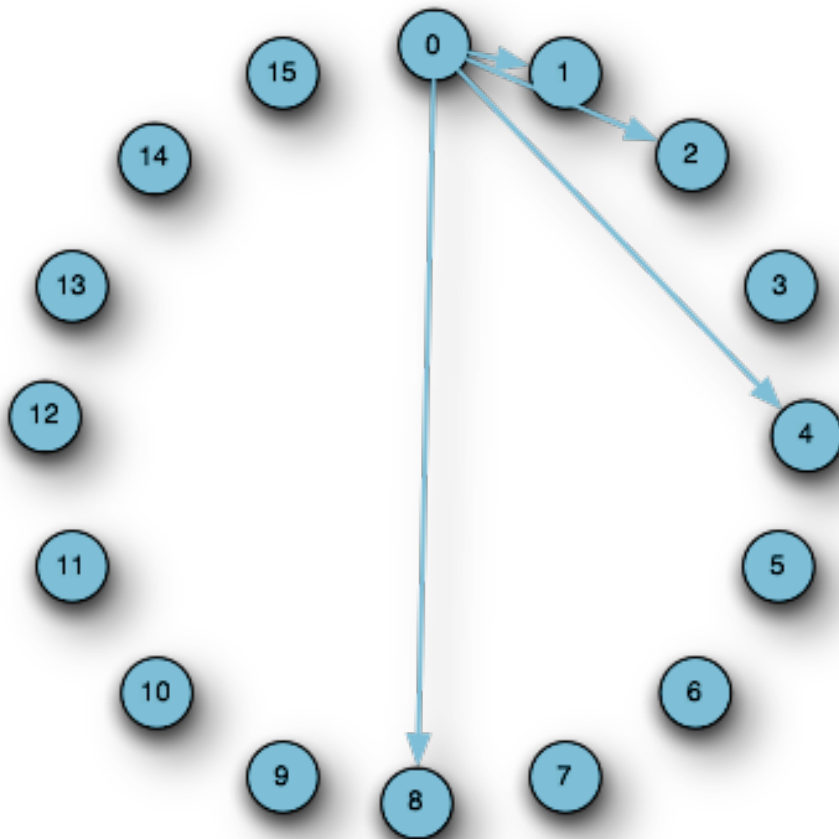


A \leftarrow 6,4

C \leftarrow 5,1,3

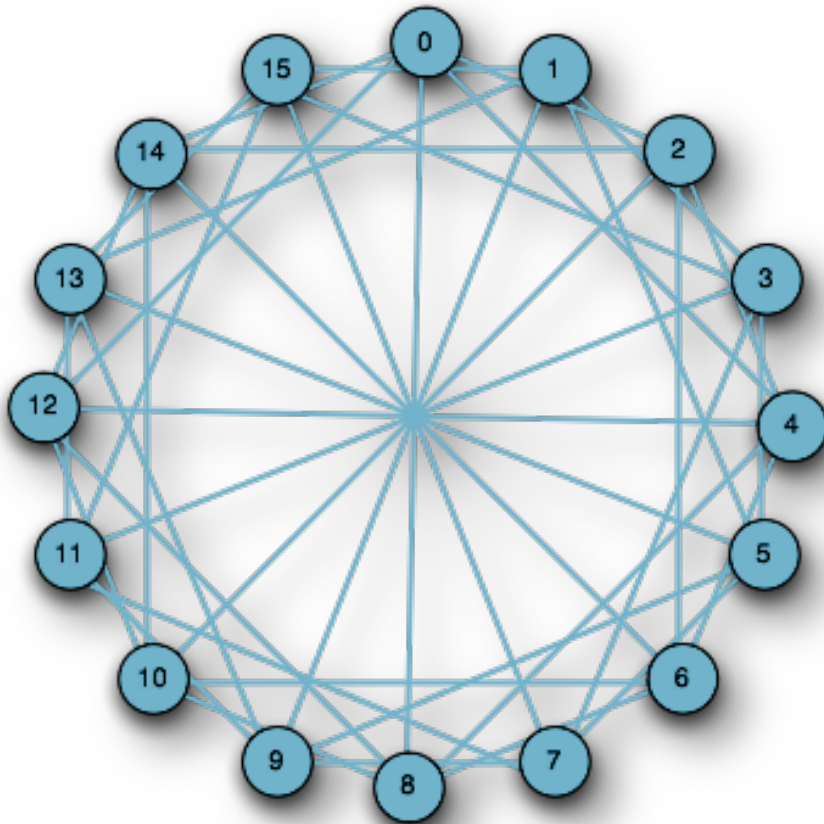
B \leftarrow 2

Chord routing



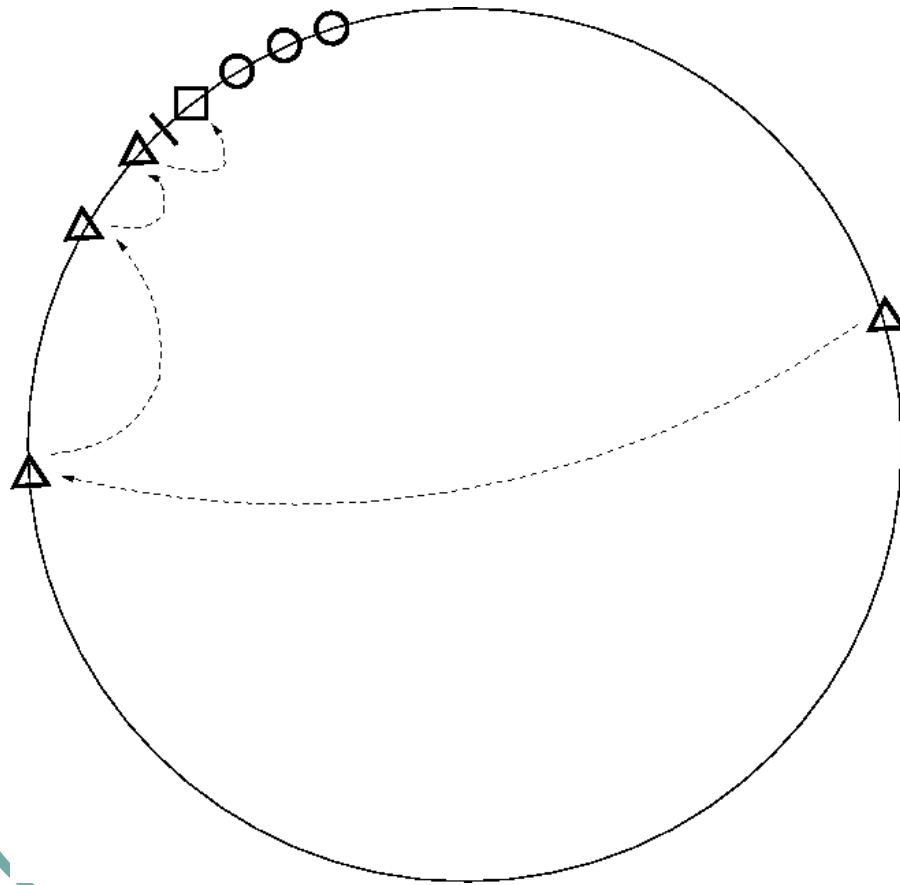
- Routing using geom-progression “fingers”
- It takes $\log(N)$
- Routing tables also $\log(N)$
- Each node keeps track (agressively maintains) only its neighbors

Chord routing



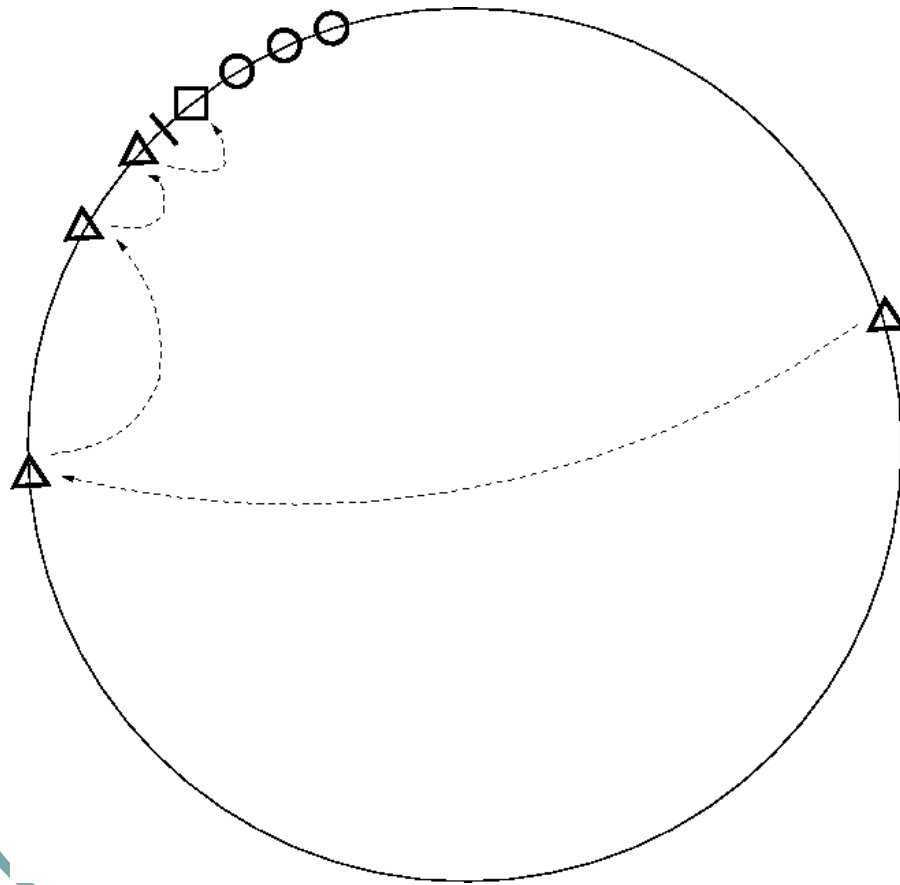
- Routing using geom-progression “fingers”
- It takes $\log(N)$
- Routing tables also $\log(N)$
- Each node keeps track (agressively maintains) only its neighbors

Chord routing



- Routing using geom-progression “fingers”
- It takes $\log(N)$
- Routing tables also $\log(N)$
- Each node keeps track (agressively maintains) only its neighbors

Chord routing assumptions

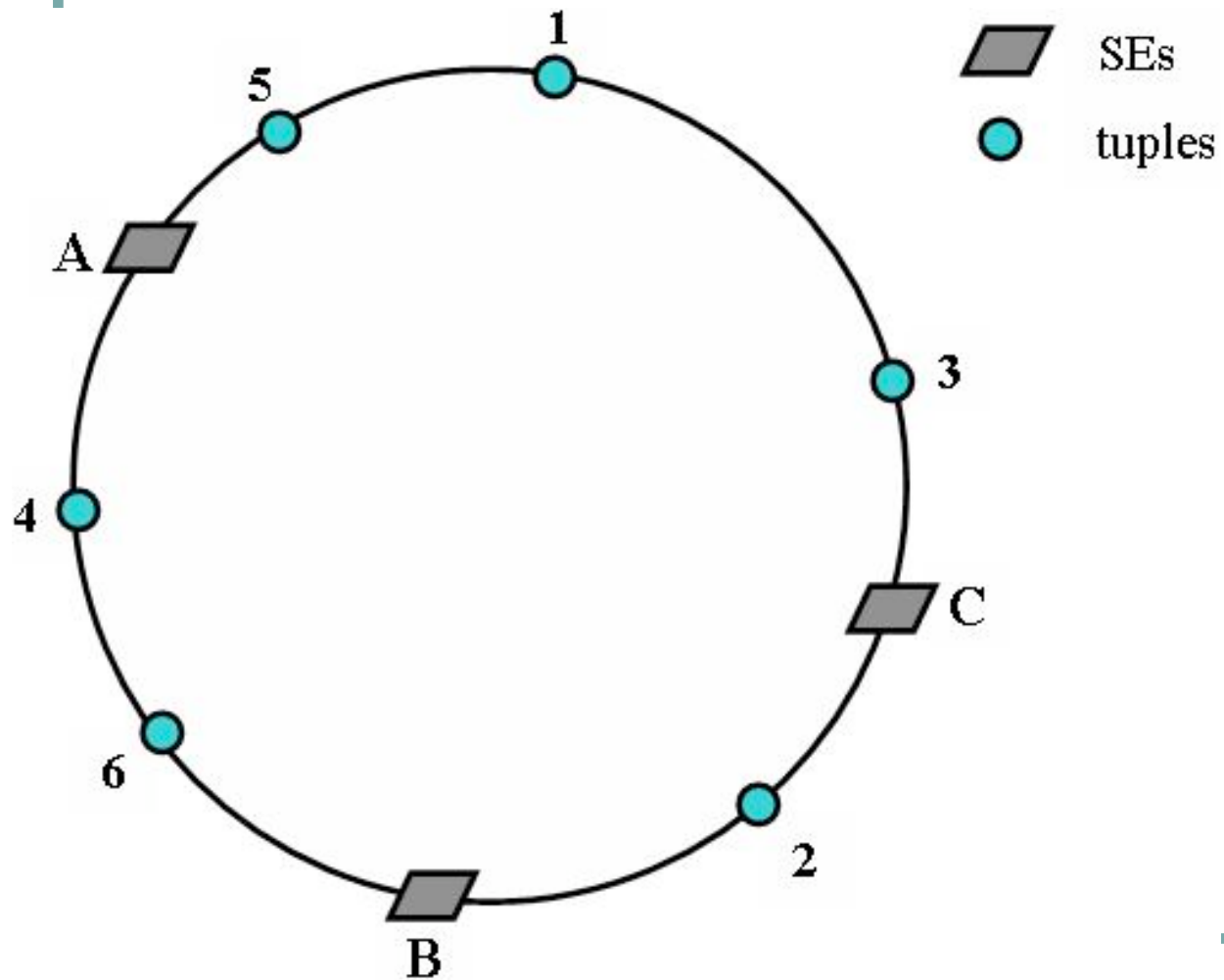


- Membership changes fast
- Updates/queries are $\log(N)$
- Could be too slow for some apps ?
 - Hopcount may matter inter-CRS, especially on WANs

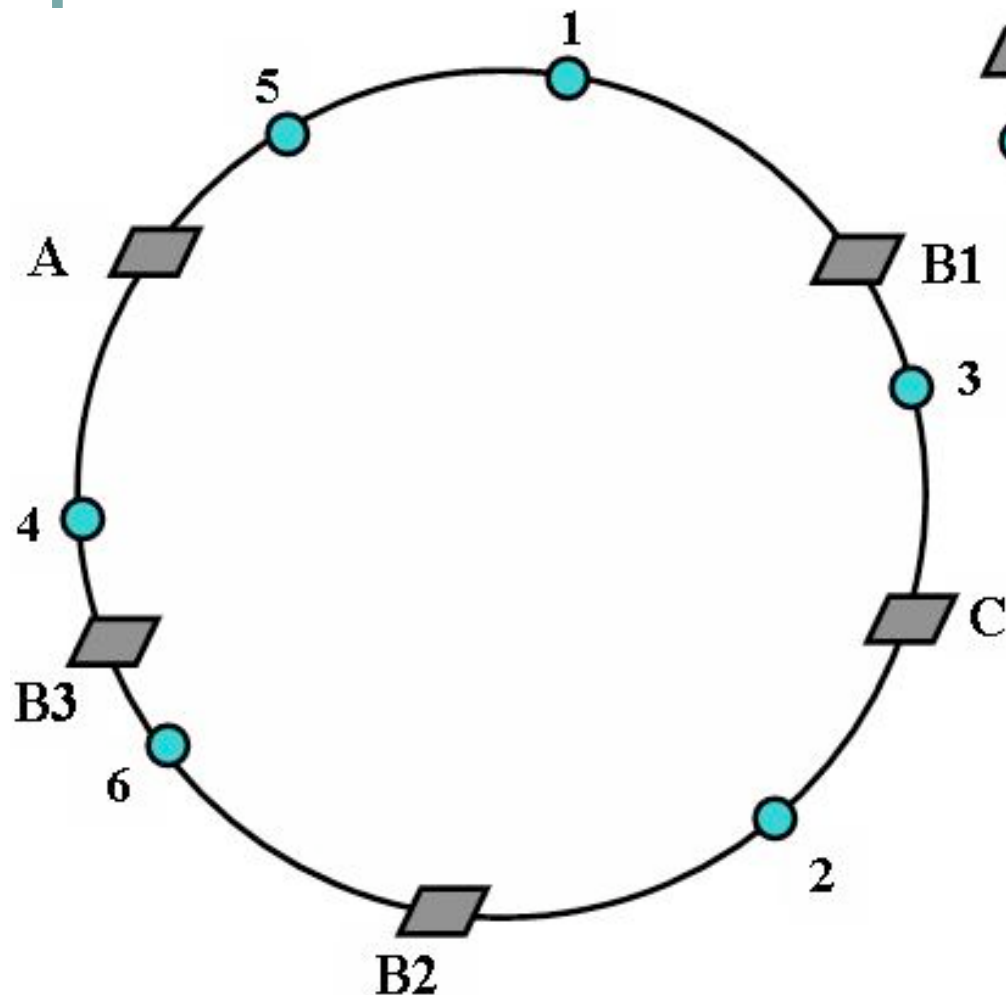
Usual Improvements

- Replace each physical SE node by many virtual nodes
=> better LB as nodes fail/join

Virtual nodes



Virtual nodes [2]



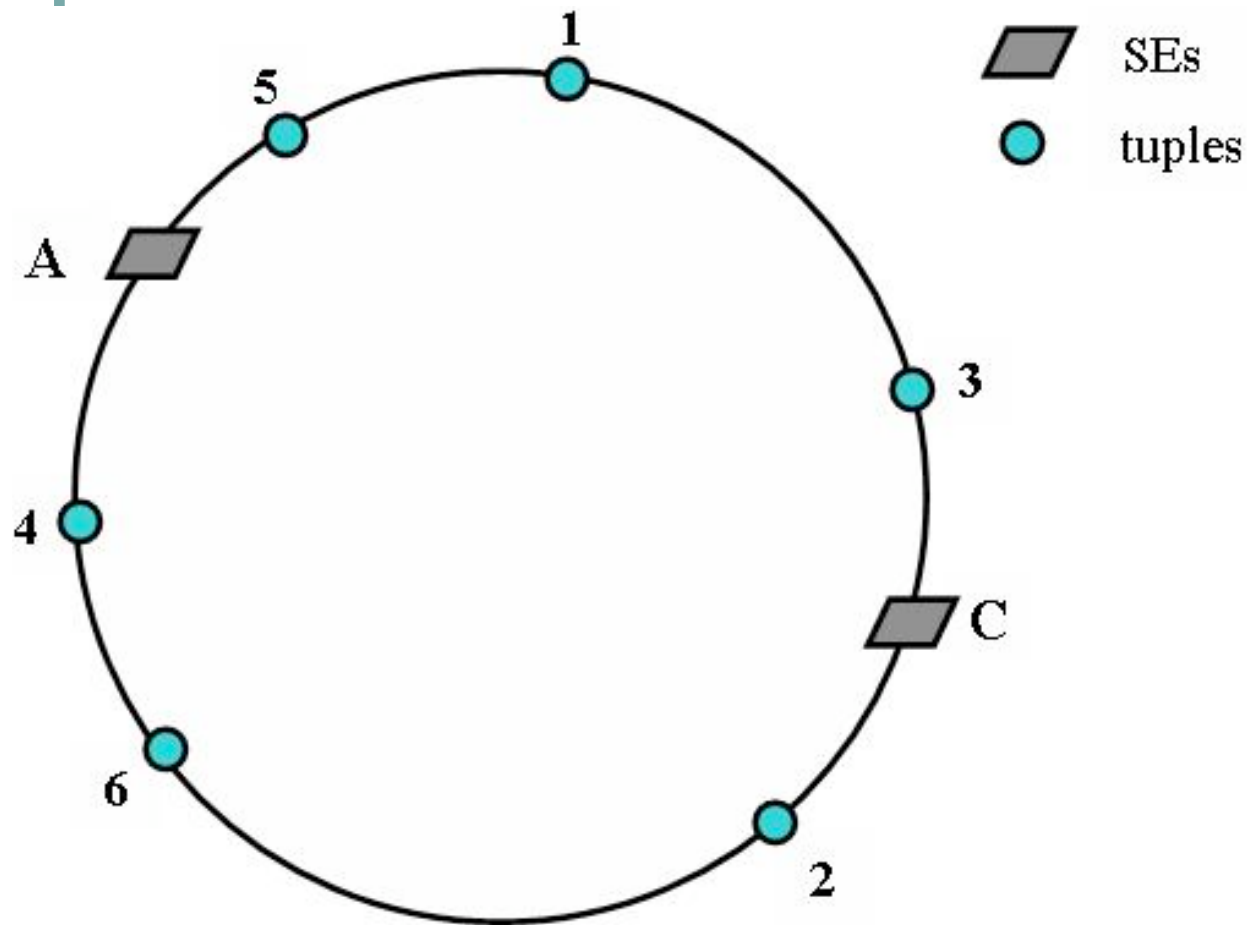
SEs



tuples

- Physical SE
=> many
virtual SEs
- B -> B1, B2, B3
- As B joins/
fails, it
changes
hands with
several SEs
- Better LB

Virtual nodes [3]

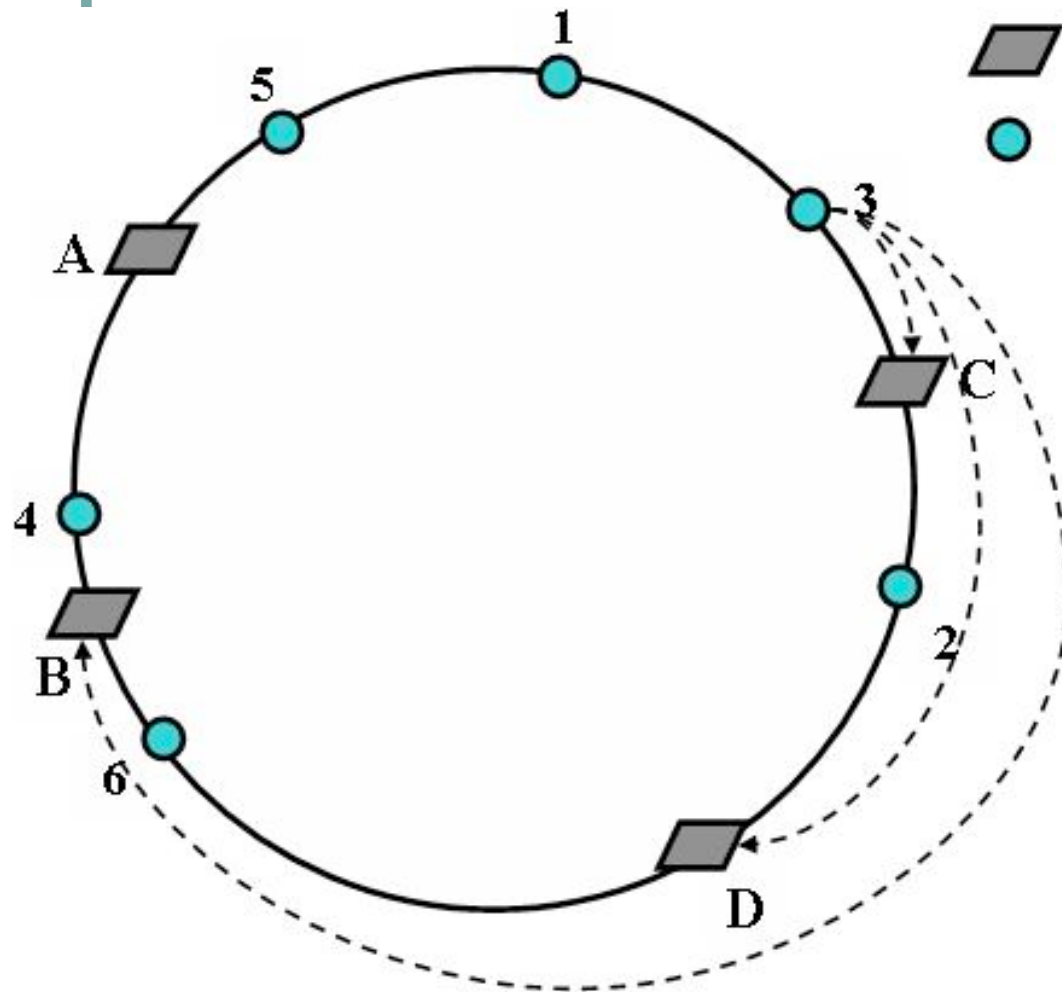


- Basic scheme:
B→A
- Virtual nodes:
B→A
B→C
- Can model SE heterogeneity
(Since it's RAM, can be quite dynamic)

Usual Improvements

- Replace each physical SE node by many virtual nodes
 - => better LB as nodes fail/join
- Replicate tuples on following K nodes rather than just 1
 - => FT

Replication



- 3 replicated on C,D,B
- 2 on D,B,A
- For 3, replica updates pushed to the others (D,B) at write-time
- If a SE fails, a tuple can easily be found on the next node CW



Adaptations for the router setup

Addressing limitations – fact sheet

- Latency optimizations:
 - Store & lock everything in RAM; prevent swapping; direct I/O; port to embedded environment
 - 1-hop routing
- Write intensive, consistency
 - Sequential consistency protocol: (Fast) Paxos
- Leverage multicast
 - Especially on group membership
- Complex queries
 - event dissemination (e.g. multicast) or indirection indexes

Memory usage

- Storage backend holds everything in in-RAM BerkeleyDBs (malloc+shmем)
- Chord design: 3 daemon processes*, DBs such as stored objects, keys, nodes now shared through shmем
 - * location, replica maintenance, DB backend
- FS => locked-in SHMEM, Linux buffer caching disabled on DB data (direct I/O)

Memory usage [2]

- Using shmem consistent with design of other router applications, as memory = valuable resource
- Local DBs can in principle be accessed by other outside processes
- Backend can be integrated with existing efficient shmem systems

Memory usage [3]

- Setup: each physical node i (e.g. CRS line card) configured to donate C_i storage capacity (RAM)
- Total storage capacity: $(\sum C_i) / K$, K is replication factor
- Stored data can be made durable over $K-1$ random failures; with serial consistency, writable over $K/2 - 1$ failures

Memory usage [4]

- C_i variable, to reflect heterogenous SEs
- Implemented via VNs
- A nice thing to do: make this variable to input free mem sensor (e.g. from CM)
 - Dynamically load balance storage wrt available mem on SEs

Embedded optimizations

- Mlock BDBs, parts of adbd, maintd
- Disable OS caching (O_DIRECT), tune BDB cache sizes (mem footprint vs speed)
- Simplify & reduce number of DBs (e.g. no expiration metadata), reduce # of servers per node
- Replace expiration mechanism by explicit deletes, unsupported in Chord
- Dependency upgrades & perf. tuning, cross-compilation to the CRS
- Upcoming: get rid of client altogether

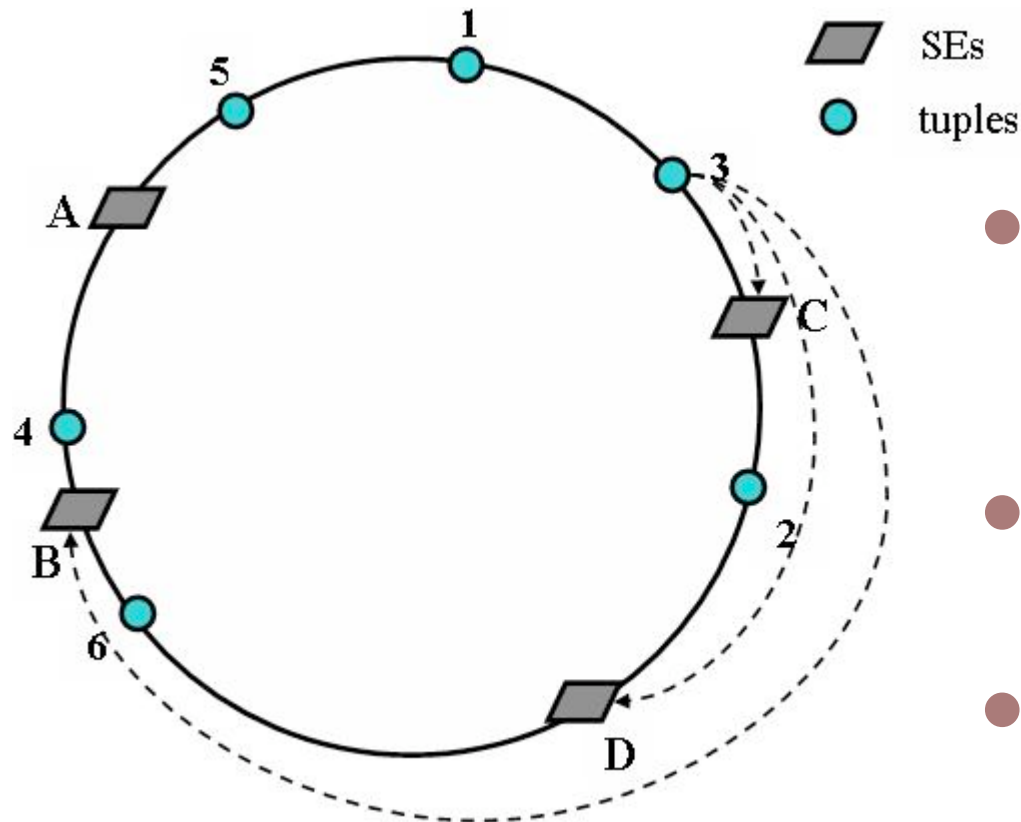
Integration with CM

- CM allows apps to be monitored and restarted (among other things)
- When failed, DHT nodes:
 - loose all data (RAM)
 - rejoin DHT fresh using dynamically discovered live bootstrap
 - take over different data load from other live nodes

Integration with CM [2]

- Between CM restarts, DHT re-replication ensures **constant replication factor K** for any tuple (i.e. any $K-1$ components can fail simultaneously)
- CM restarts ensure **constant number of SEs** over time
- MTTR given by DHT heartbeat + repair w.r.t. data and by CM failure detection + DHT node restart time w.r.t. # SEs

Replica synchronization



- Consistent order of updates (serializability) ?
- Serializability can be achieved by e.g. RW quorums
- But such updates need to be atomic
- Solvable by distributed consensus

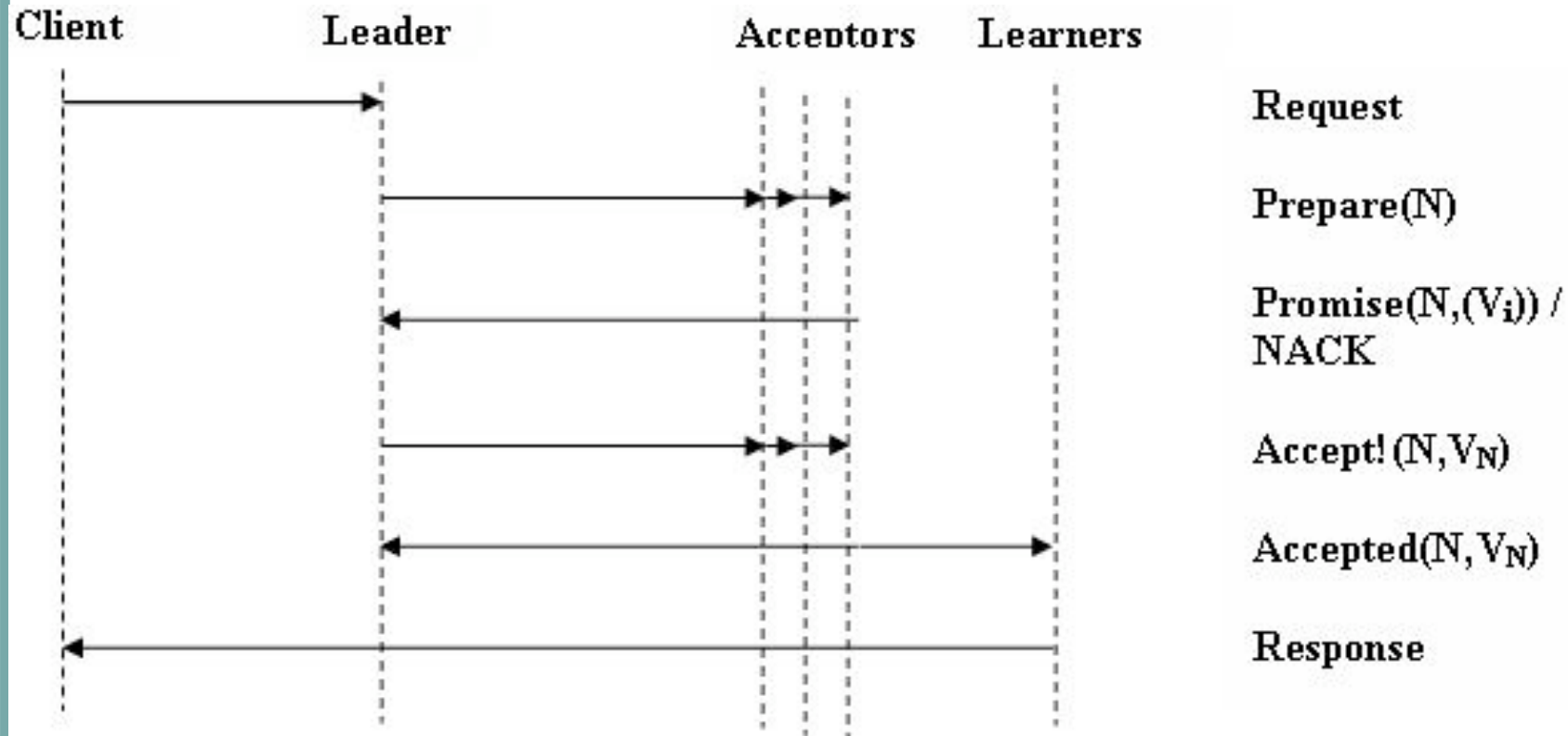
Paxos factsheet

- Alg. to achieve *distributed consensus*
- E.g. have several replicas agree on the order of operations applied to them
- Non-blocking, safe in face of:
 - regular acceptor failures, leader failures, multiple dueling leaders, lost/duplicate/reordered packets, arbitrarily slow nodes

Paxos factsheet [2]

- Can be made pretty fast (\gg 3PC) –
Fast Paxos:
 - Collision-free case \rightarrow 2 X 1-way delays
 - Duel case (sh'd be infrequent in our setup) \rightarrow 4 X 1-way delays most often
- Progress made if 1 single leader remains in charge long enough
- Used in Google Chubby, several distributed DB projects

(Basic) Paxos



Paxos with Chord

- Client is node that initiates write
- Elected leader for a given query is tuple successor node
- Lookup inconsistencies lead to dueling leaders, solved by Paxos
- Issue: Leaders/Clients need to agree on replica membership
 - Can be made improbable with high quorum
 - Can be guaranteed with consensus on replica membership

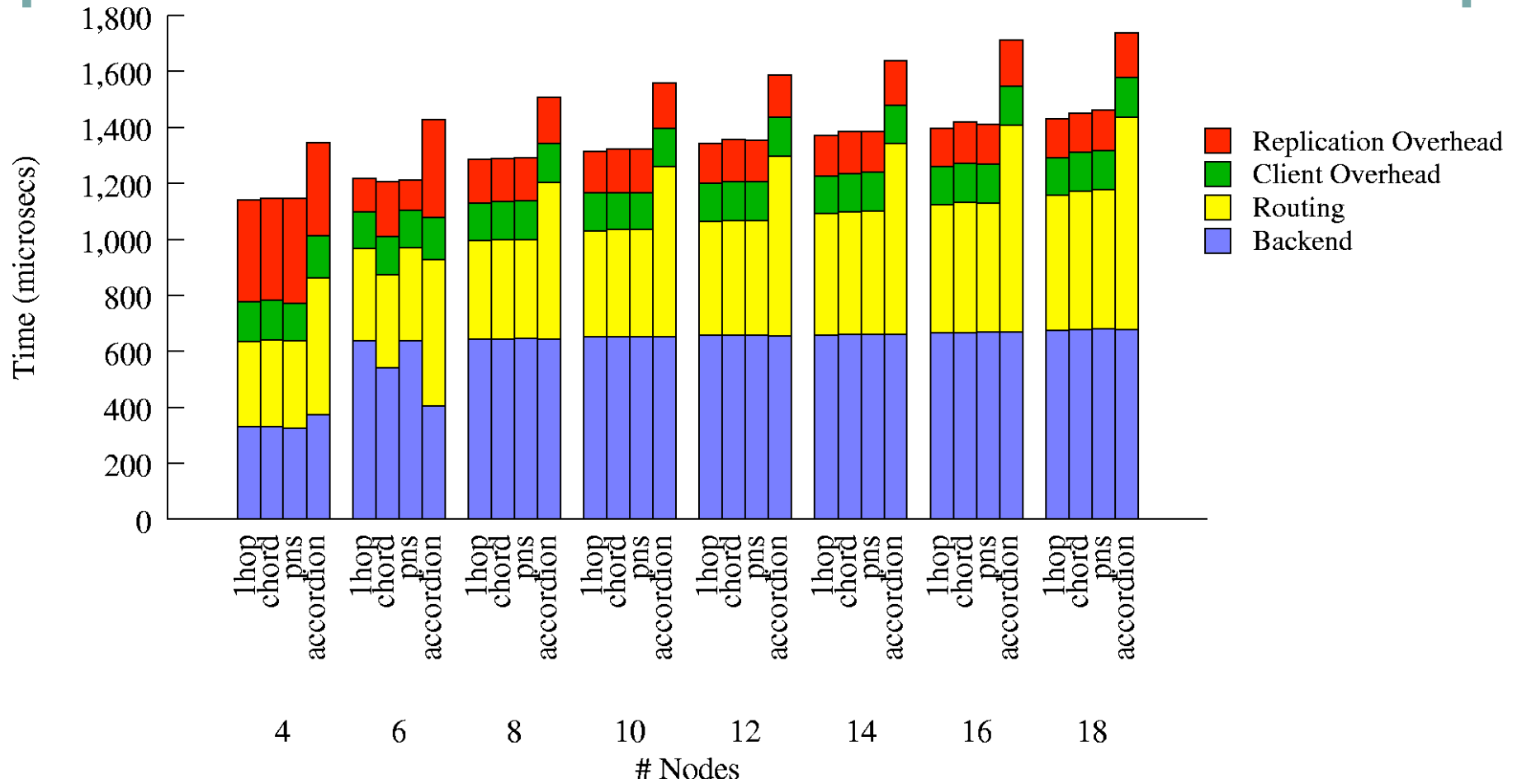
Routing

- Integrated with Chord routing, which includes classic Chord (iterative & recursive geom progression) and Accordion (harmonic small world fingers)
- We do 1-hop routing via full routing caches
- Based on requirements (e.g. intra- or inter- PoP, router scale, RAM), routing alg. can be toggled at run- (=DHT startup) time

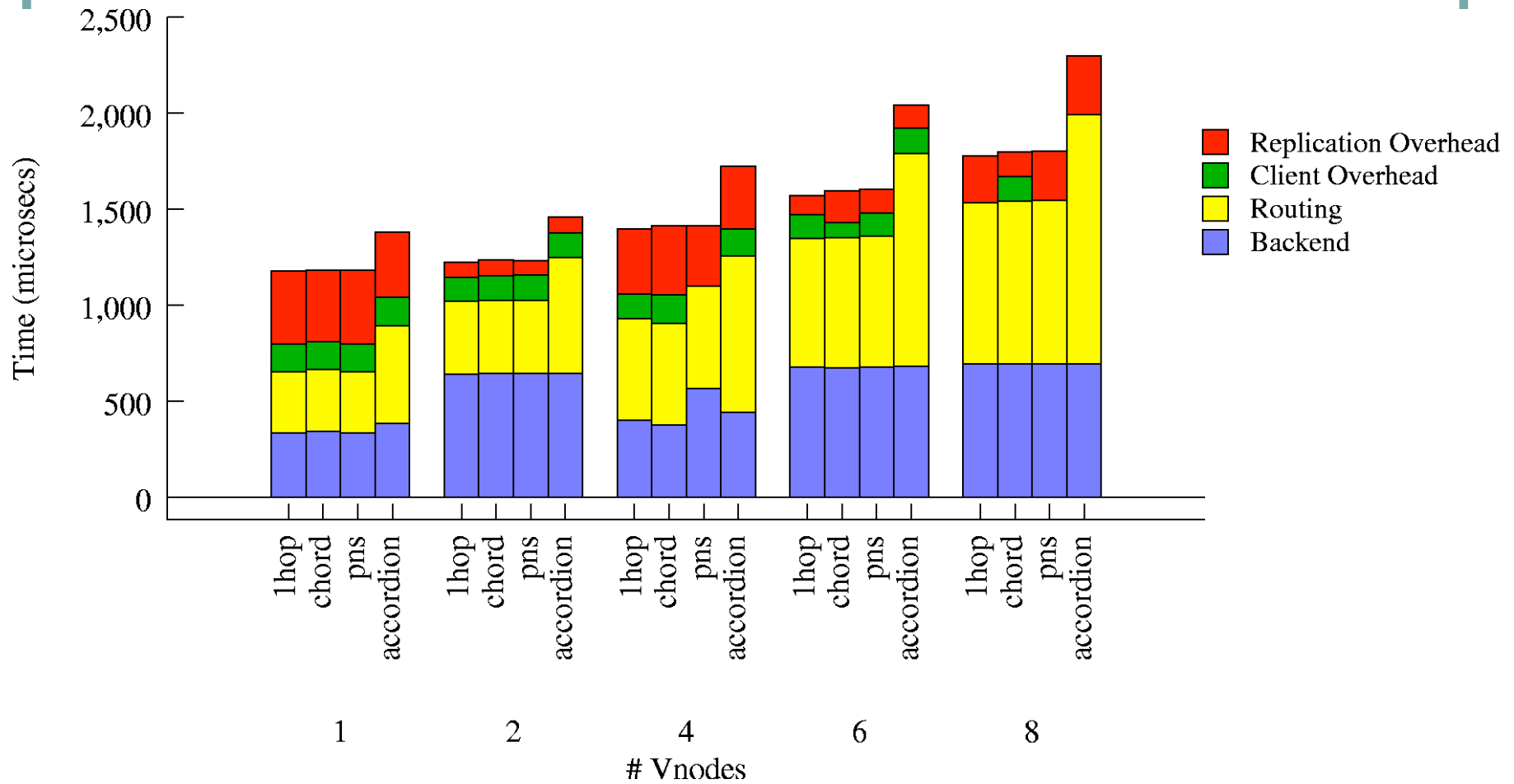
Platforms

- Ludd-1: 18 node emulation cluster:
 - AMD Opteron 250, 2.4 GHz
 - 4 GB RAM
- Half-q CRS, usable: 2 RPs, 3 LCs
 - RP: PPC 7447, 1.2 GHz, 4 GB RAM
 - LC: PPC 7455, 1 GHz, 2 GB RAM

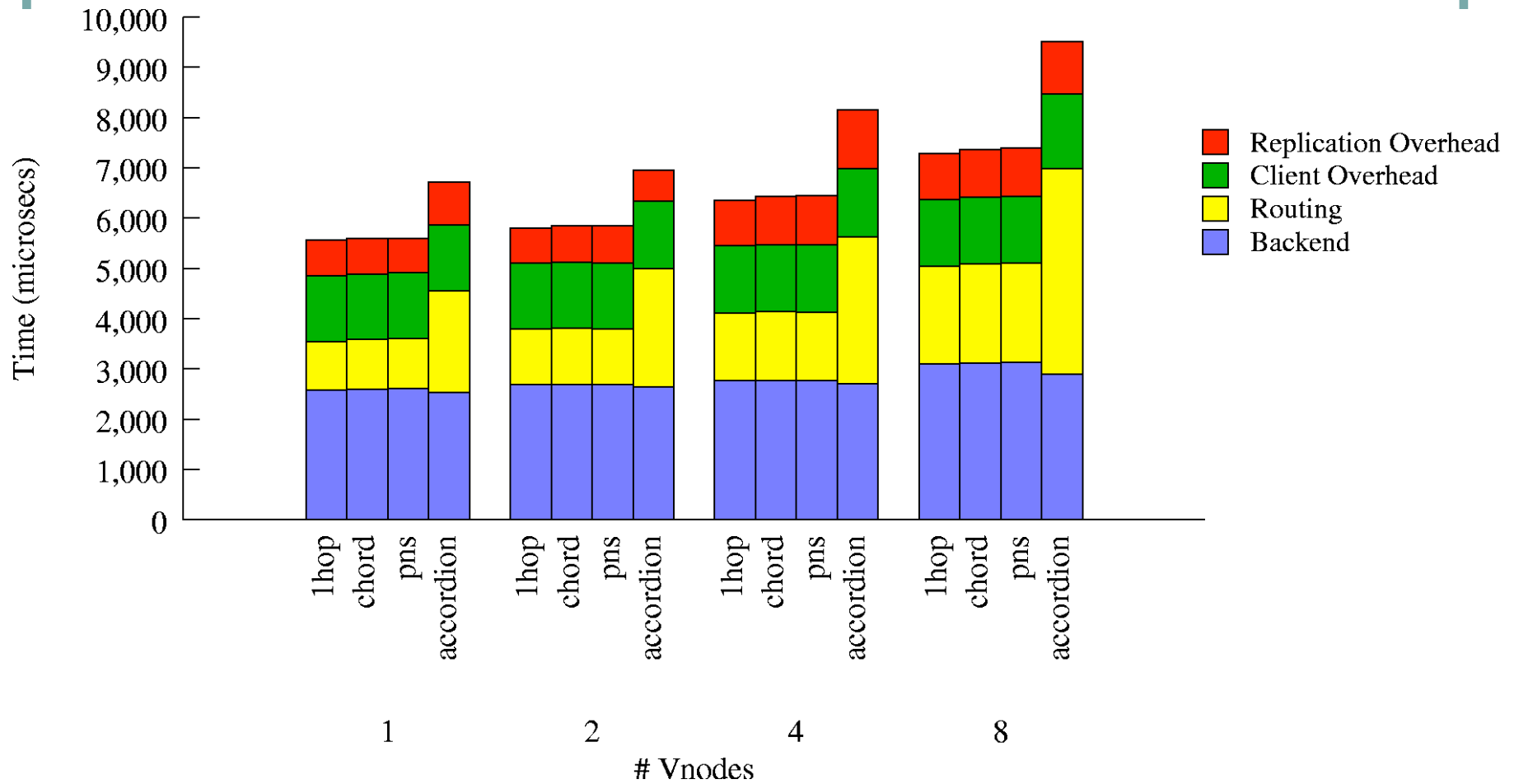
Scalability in # SEs : LUDD-1



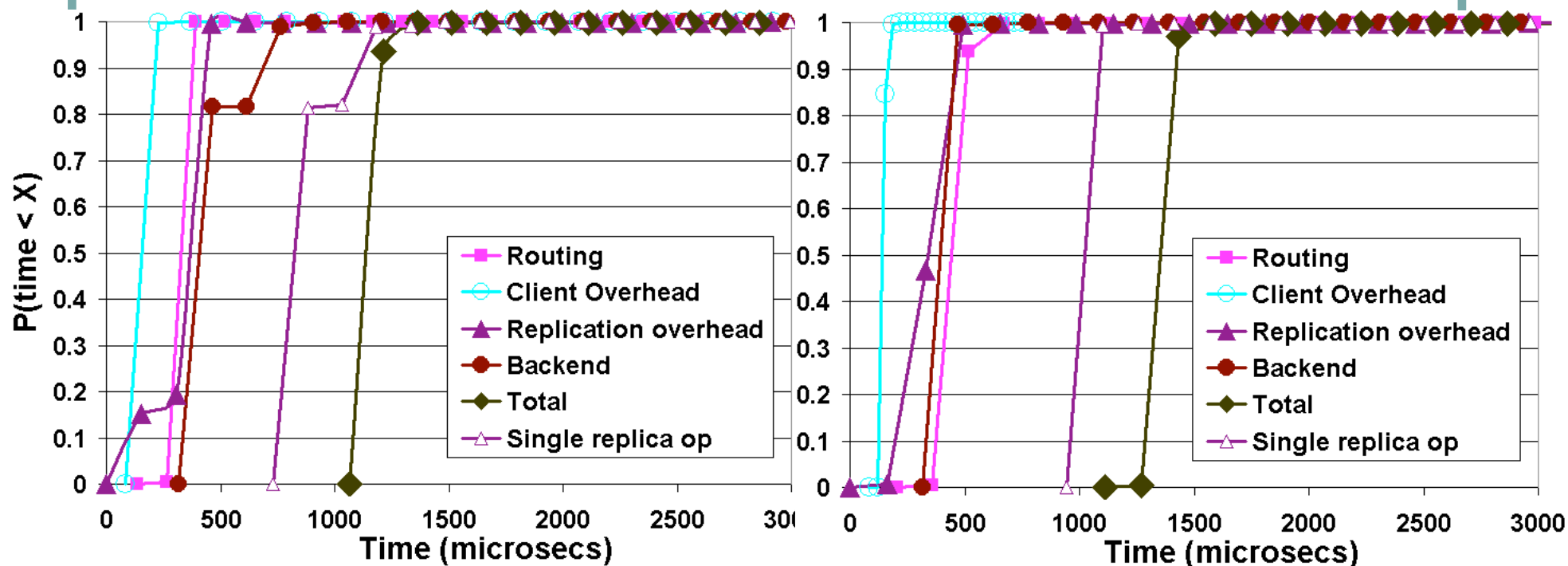
Scalability in # Vnodes : LUDD-1



Scalability in # Vnodes : CRS



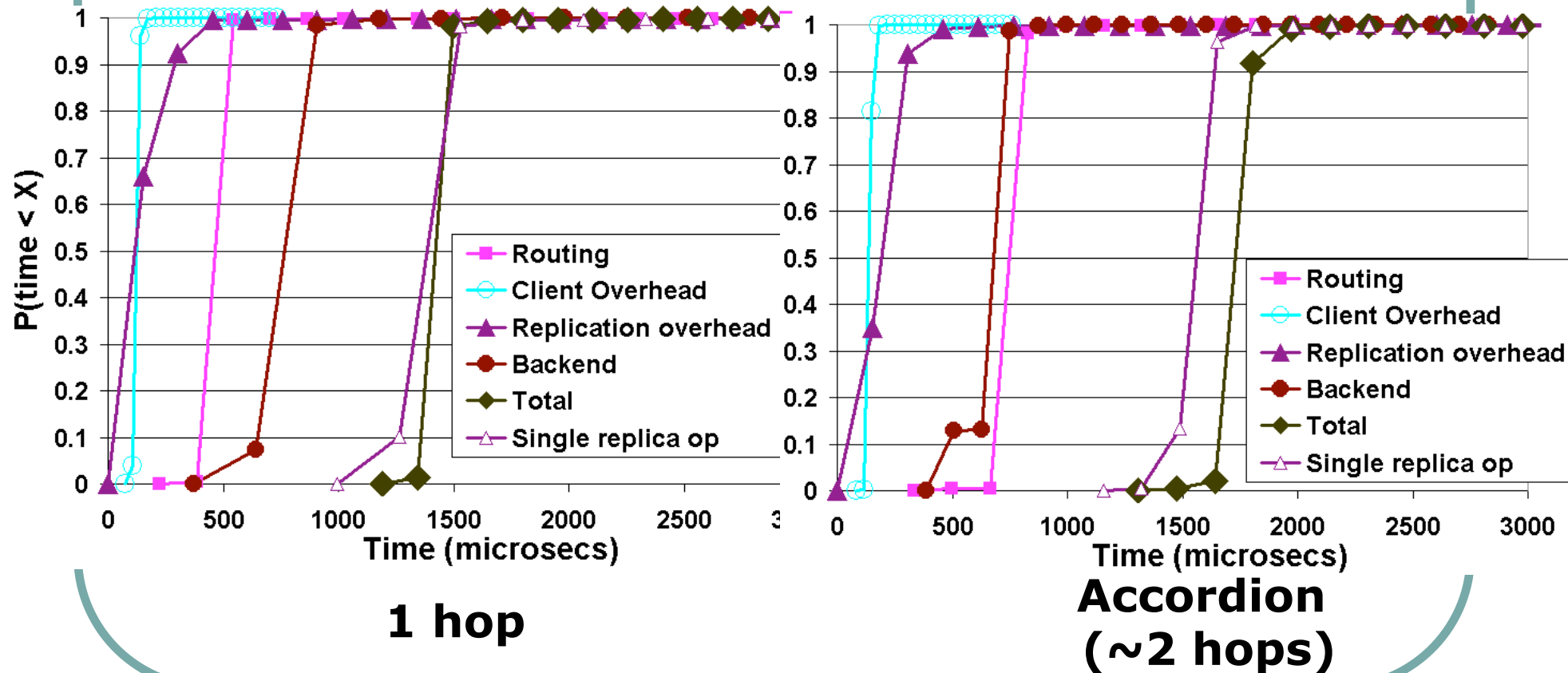
Distribution of times (LUDD1, 4 nodes)



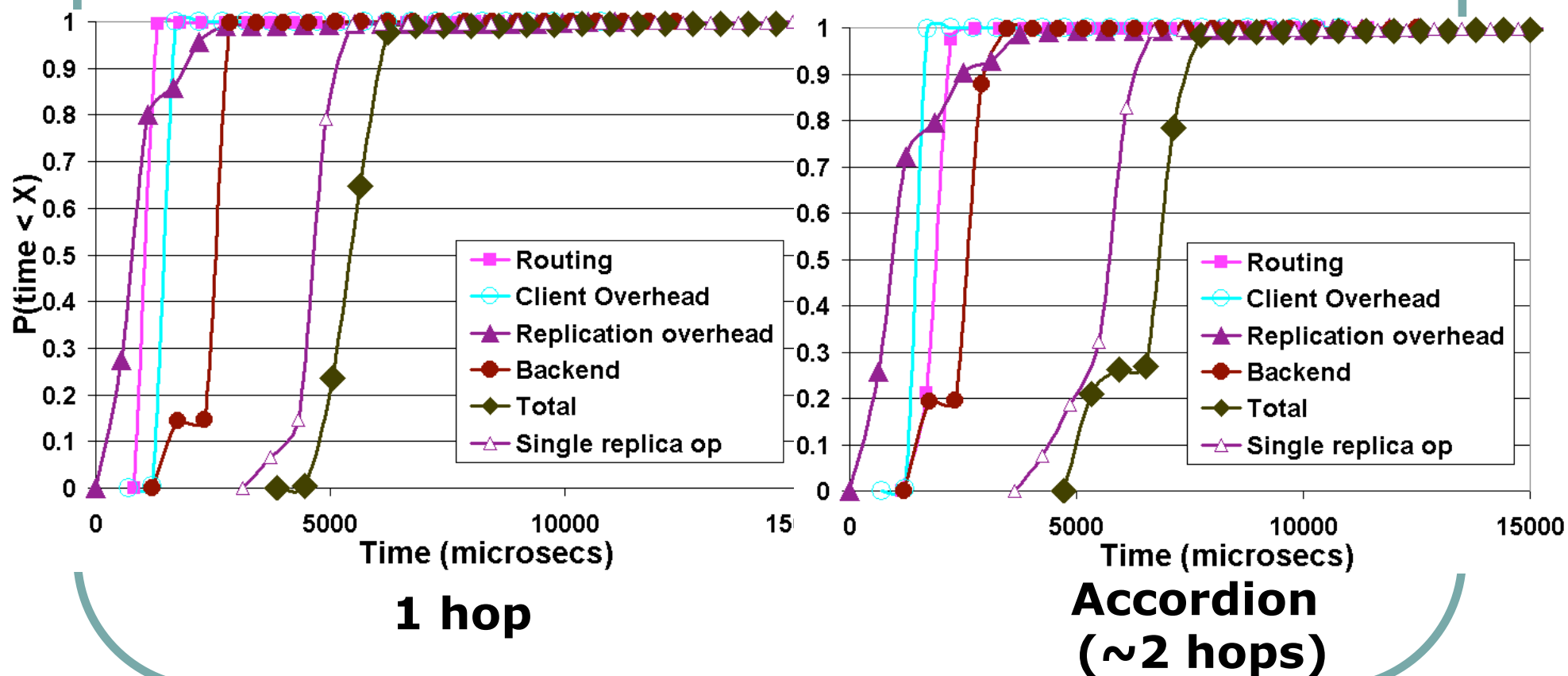
1 hop

**Accordion
(~2 hops)**

Distribution of times (LUDD1, 18 nodes)



Distribution of times (CRS, 4 nodes)



Related work (list non-exhaustive)

- Plethora of work on log-hop, $O(1)$ - and 1-hop DHTs, including aspects:
 - $O(1)$ -hop: Kelips, r-Kelips, [Leong et Li 04] (1 hop), ...
 - Replication & r. maintenance: Carbonite, Beehive ...
 - Range queries: r-Kelips again, SkipNet, PHT, [Awerbuch 03], ...
- Within datacenter community, e.g. Dynamo
- For consistency, close to our context:
 - Work by S. Haridi
 - Chain replication
 - PaxonDHT proposal (which doesn't handle replica membership changes though)
 - By L. Lamport, recently: Vertical Paxos

Related work [2]

- “Transactional memory”, e.g. Sinfonia; ours is malleable, more focused on FT aspects
- DHTs & routers: a recent DHT developed by group in Cisco for edge routers (e.g. SSO apps), WAN-, Pastry-based, not really latency-targeted, read-intensive, quite classic
- We target a fast, in-memory, write-intensive DHT that can leverage well-connected clusters/groups of routers; consistency and speed are key

Future Work

- Closer cluster/platform-based integration (e.g. leverage CM's dynamic multicast groups for communication with replicas)
- Free memory sensors and such
- Benchmarking/testing in several other scenarios
- Integration with IS-IS shim logging system

Future work [2]

- A good extendable framework to further support:
 - Various consistency mechanisms
 - Durability/Availability/Performance tradeoff
 - May extend to other failure models, e.g. Byzantine voting etc
 - Enable components to assume a “mutually suspicious” stance