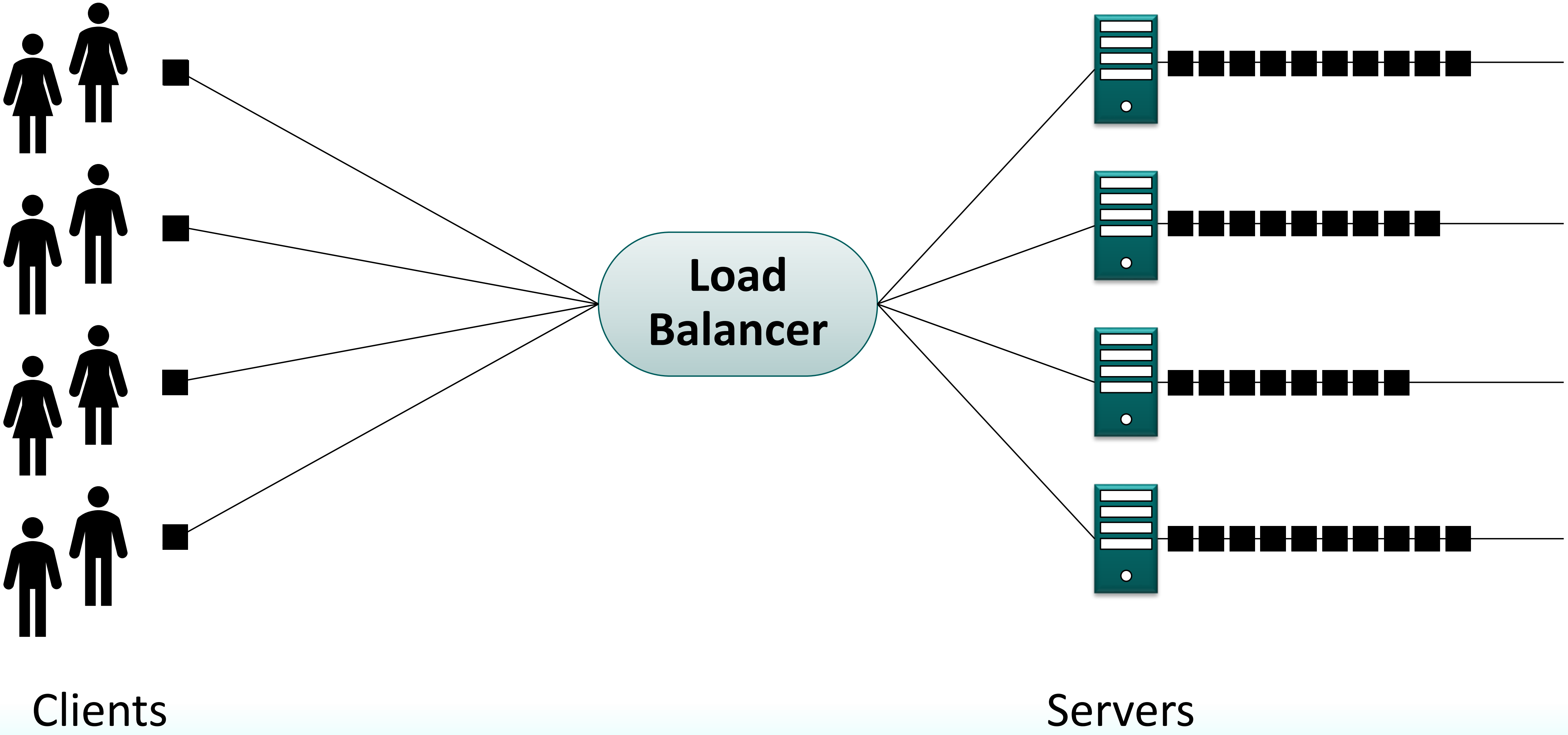
Load Balancing with JET:

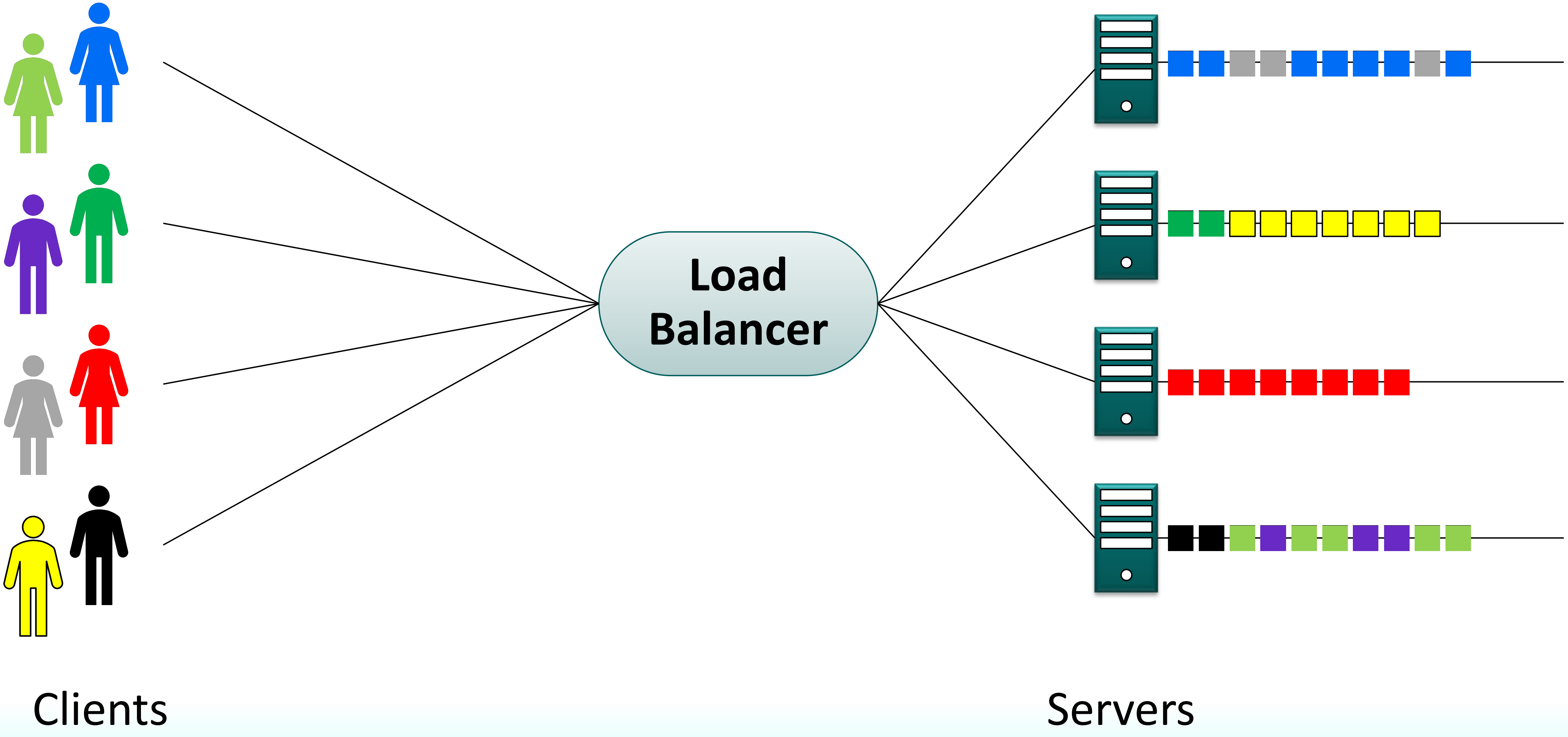
Just Enough Tracking for Connection Consistency

Gal Mendelson, Stanford
Shay Vargaftik, VMware Research
Dean H. Lorenz, IBM Research – Haifa
Kathy Barabash, IBM Research – Haifa
Isaac Keslassy, Technion
Ariel Orda, Technion

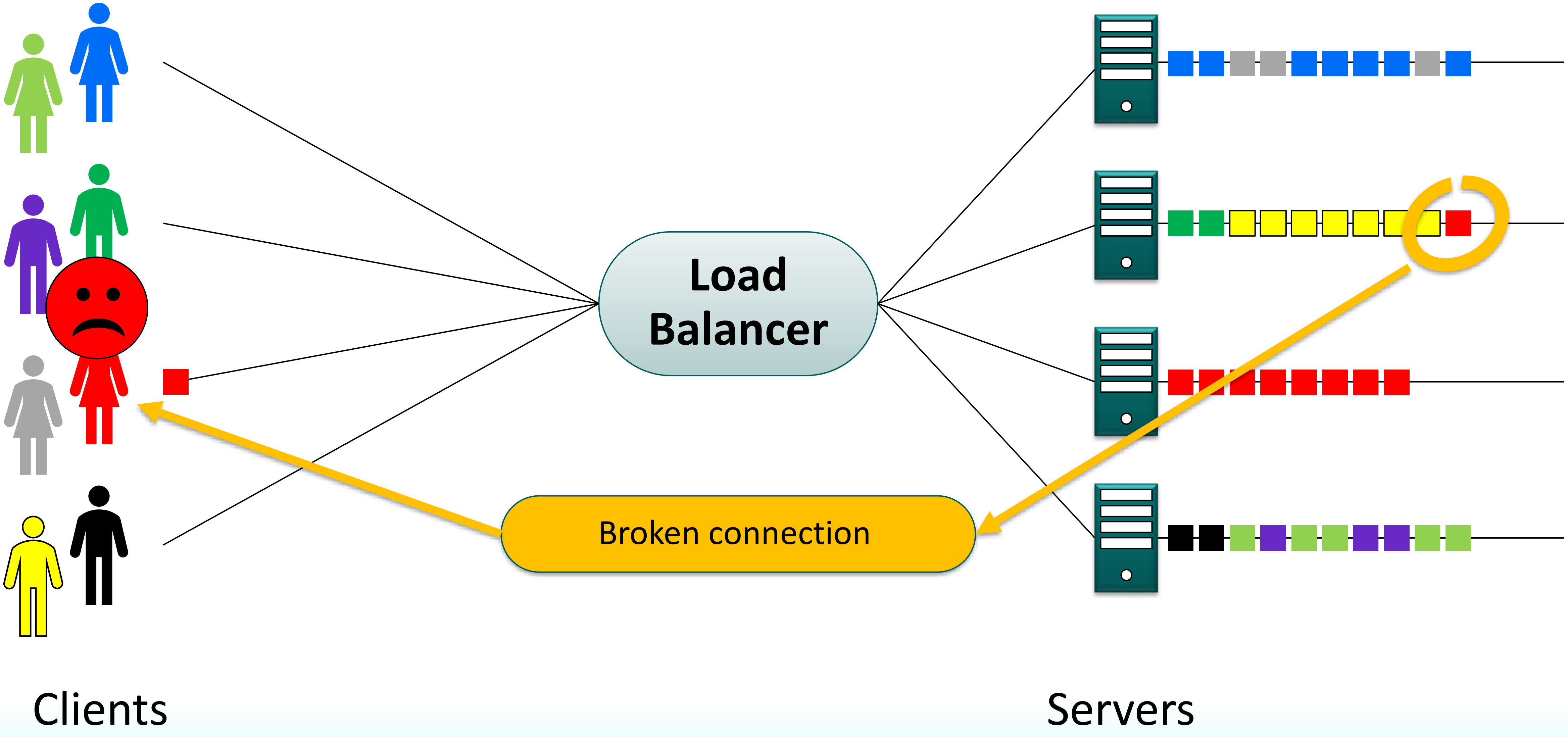
Load Balancing



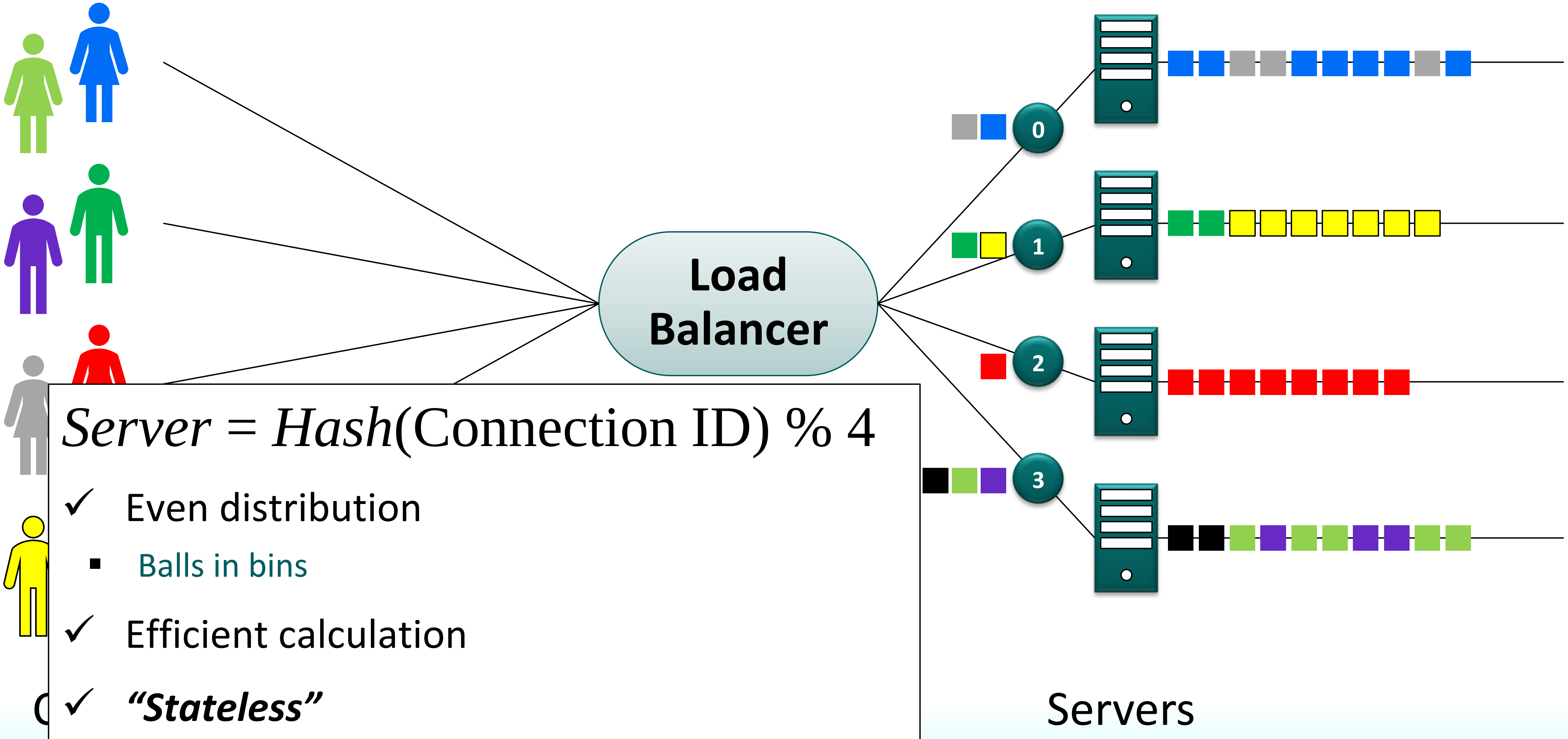
Per-Connection Consistency (PCC)



Per-Connection Consistency – PCC Violation



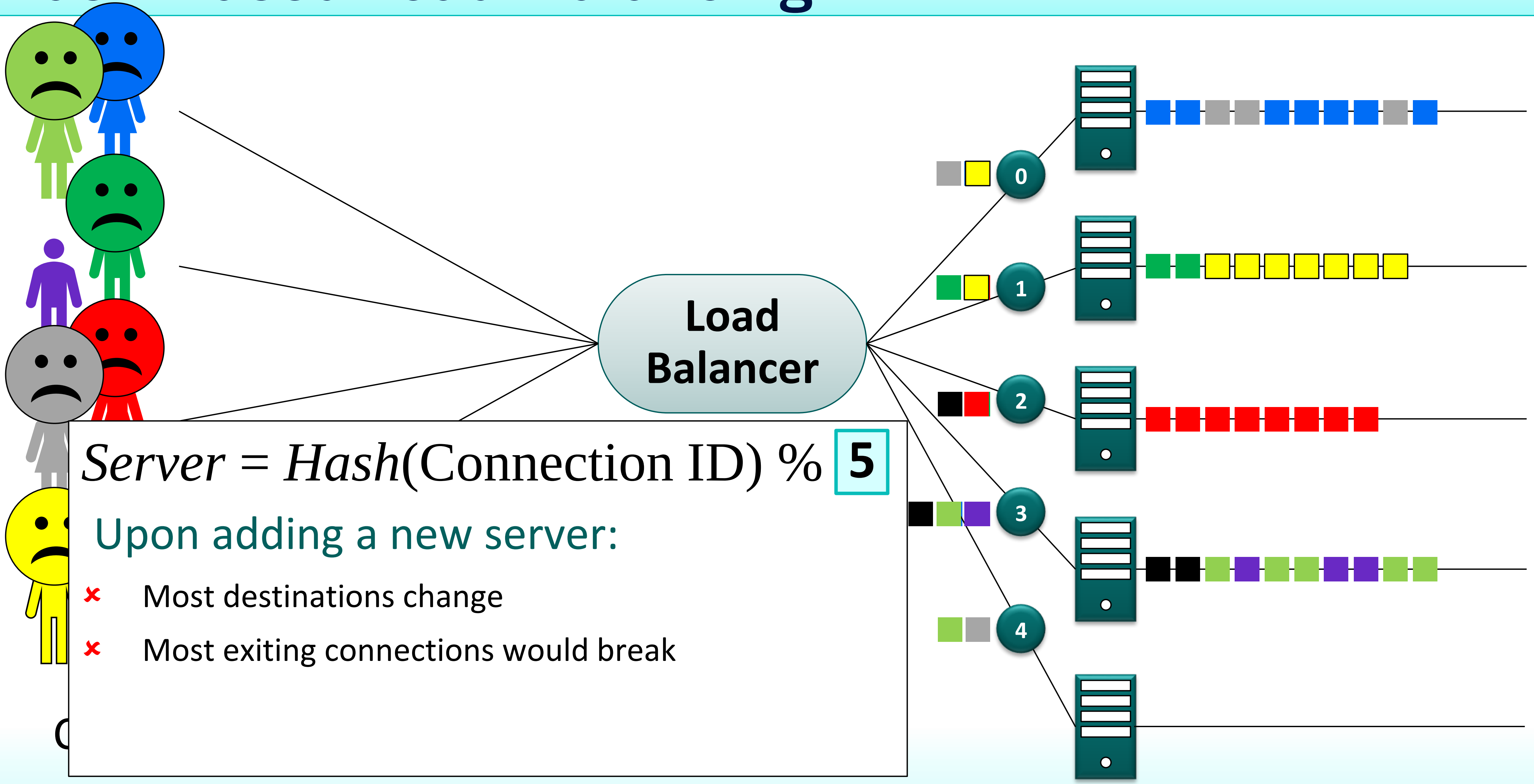
Hash-Based Load-Balancing



$$\mathit{Server} = \mathit{Hash}(\text{Connection ID})\ \%\ 4$$

- ✓ Even distribution
  - Balls in bins
- ✓ Efficient calculation
- ✓ ***“Stateless”***

Hash-Based Load-Balancing

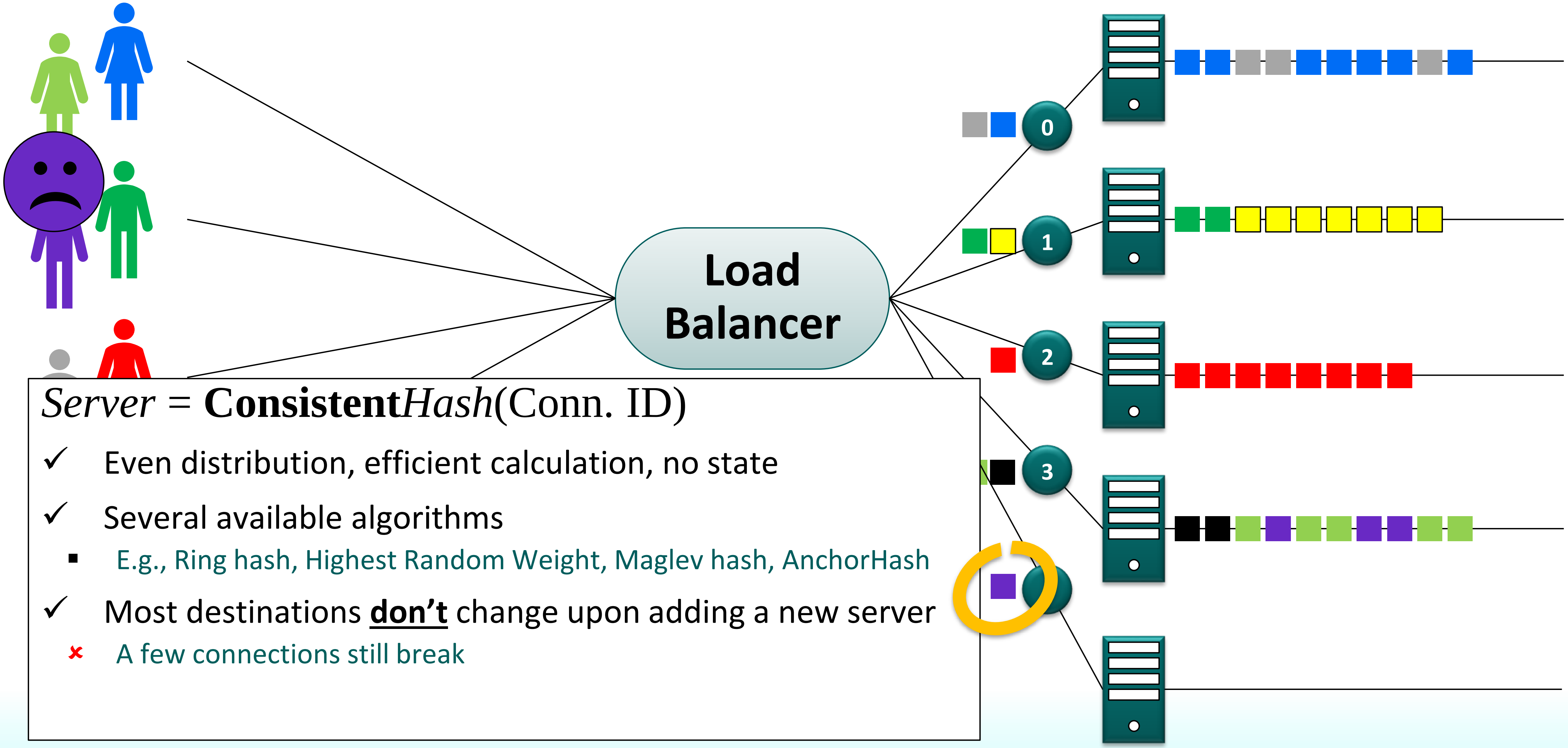


$$\mathit{Server} = \mathit{Hash}(\text{Connection ID})\ \%\ \boxed{5}$$

Upon adding a new server:

- ✘ Most destinations change
- ✘ Most exiting connections would break

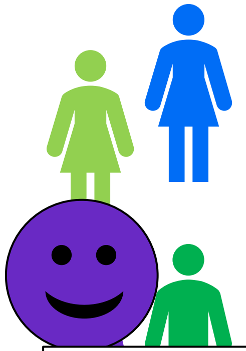
Load-Balancing with a *Consistent Hash*

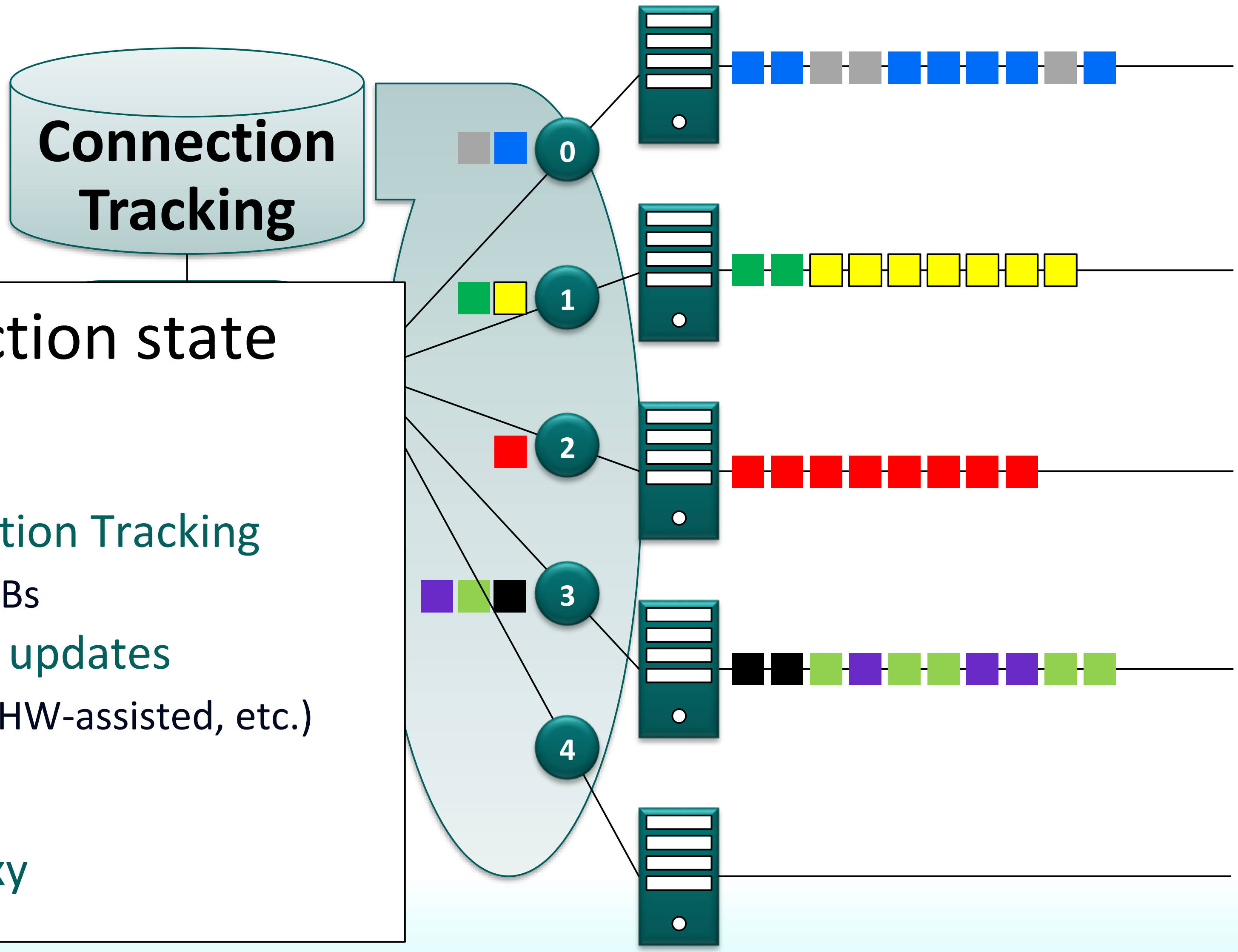


Server = **Consistent***Hash*(Conn. ID)

- ✓ Even distribution, efficient calculation, no state
- ✓ Several available algorithms
  - E.g., Ring hash, Highest Random Weight, Maglev hash, AnchorHash
- ✓ Most destinations **<u>don't</u>** change upon adding a new server
  - × A few connections still break

Stateful Load-Balancing





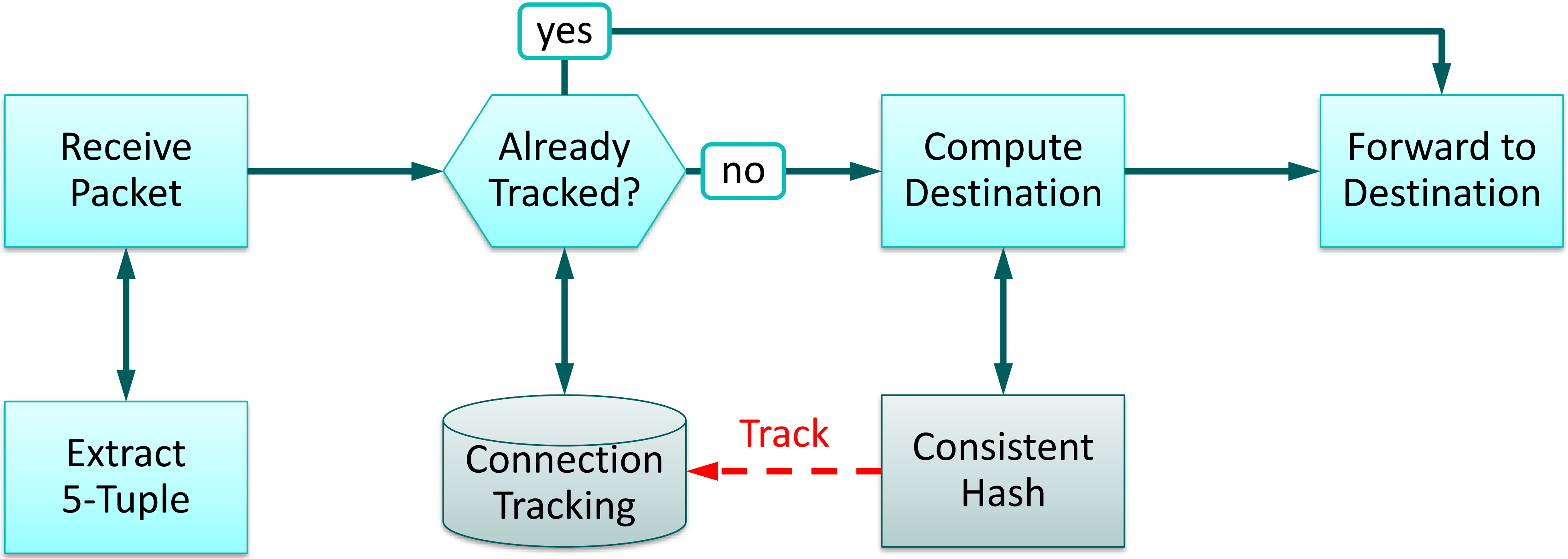
– Remember per-connection state

- ✓ Never violate PCC
  - » For the tracked connection
- ✘ Need enough space for Connection Tracking
  - » More state to sync for distributed LBs
- ✘ Need line-rate key lookups and updates
  - » Many optimizations (Bloom filters, HW-assisted, etc.)

– Used in practice

- Maglev, Katran, NGINX, HAProxy

Stateful LB Flow



“*Stateless*” Load-Balancing

– Stateful, but no state at load-balancer

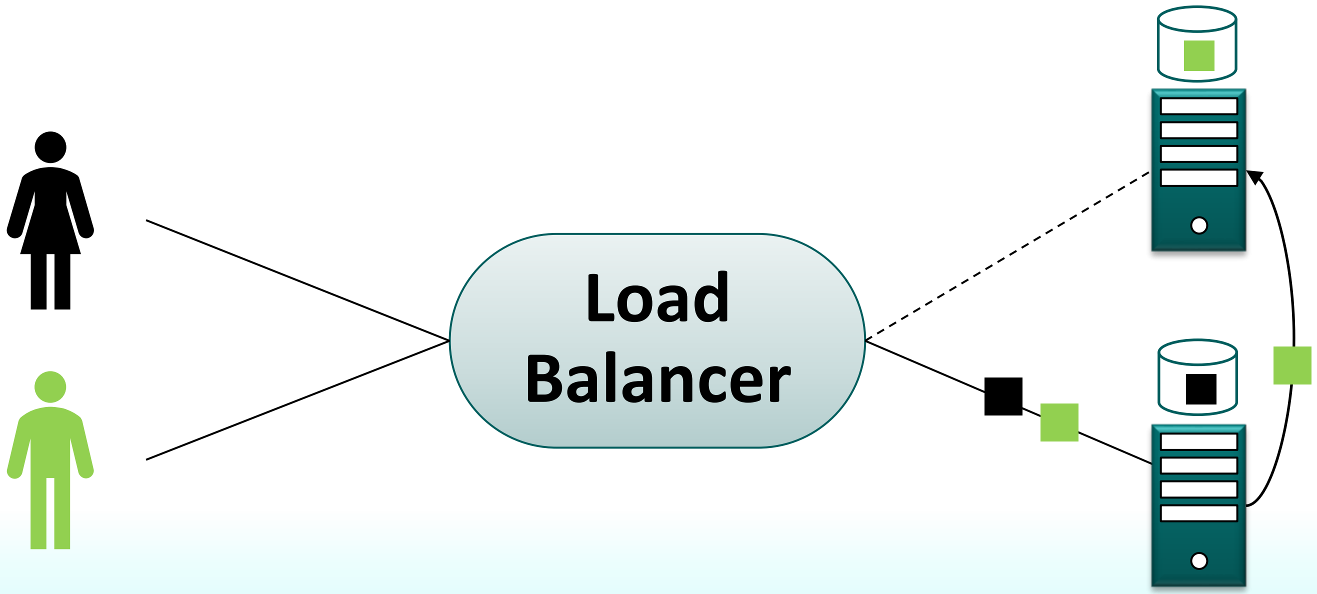
<u>For example:</u>

– State may be saved at back-end servers

- Redirect to correct server if needed
  - » E.g., Faild (NSDI ‘18), Beamer (NSDI ‘18)

– State may be saved at user

- Cookies
  - » E.g., Cheetah (NSDI ’20)



“*Stateless*” Load-Balancing

– Stateful, but no state at load-balancer

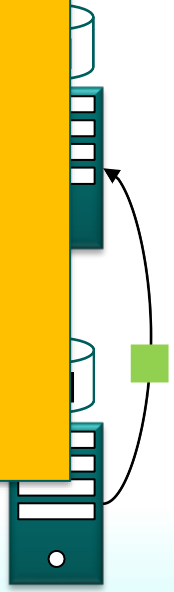
– State

- “Does
- If not,
  - » E.g., F

– State

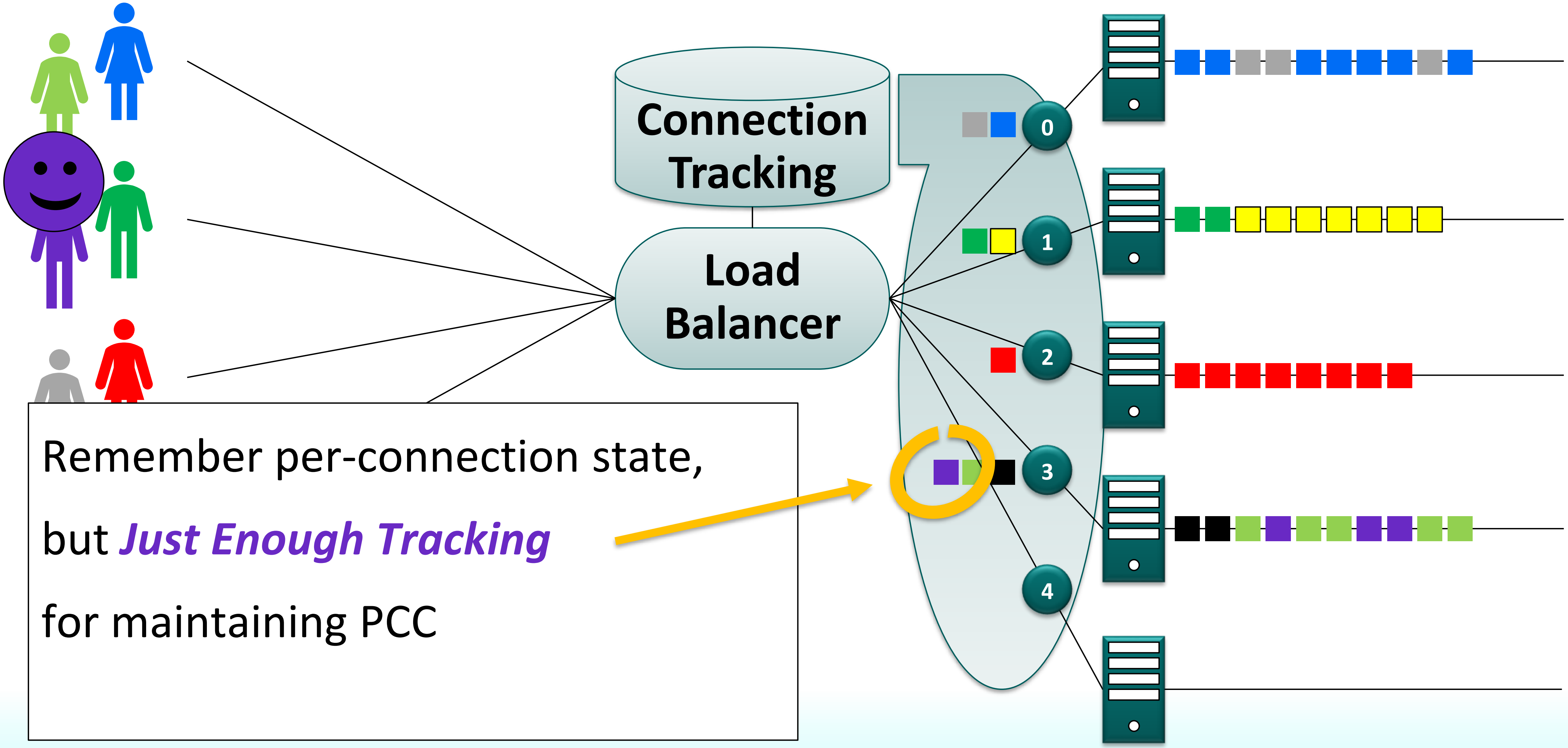
- Widel
  - » DNS r
  - » Cooki
- L4 co
  - » E.g., C

This work is about *stateful* load-balancers



served by
server #1

Stateful Load-Balancing with *JET*



Remember per-connection state,

but ***Just Enough Tracking***

for maintaining PCC

How Much is "*Just Enough Tracking*" ?

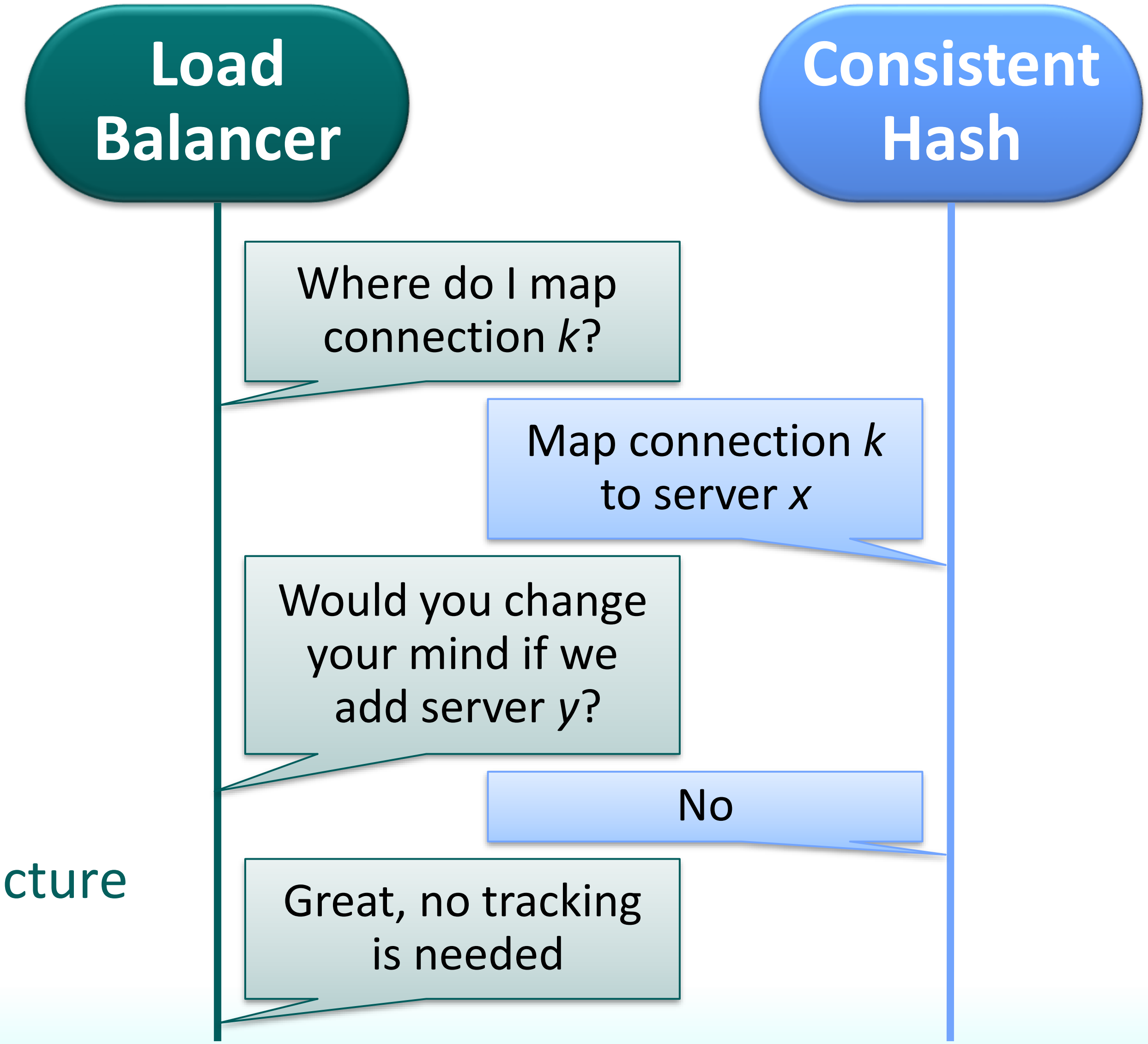
- Answer: very little ! (if you are careful)
  - Only track connections that would otherwise break
- Consistent-hashing:
  - Server **addition**
    - Only $\approx 1/N$ connections are remapped
    - These must be tracked to preserve PCC
  - Server **removal**
    - Only connections on removed server are remapped
    - These connections would break ➔ no need to track
- Tracking $\approx$*1/N* of connections is "just enough" to preserve PCC !
  - Naturally extends to multiple additions/removals
    - Tracking ~10% of connections can be "just enough" (see paper for details)

Preparing for Server Additions

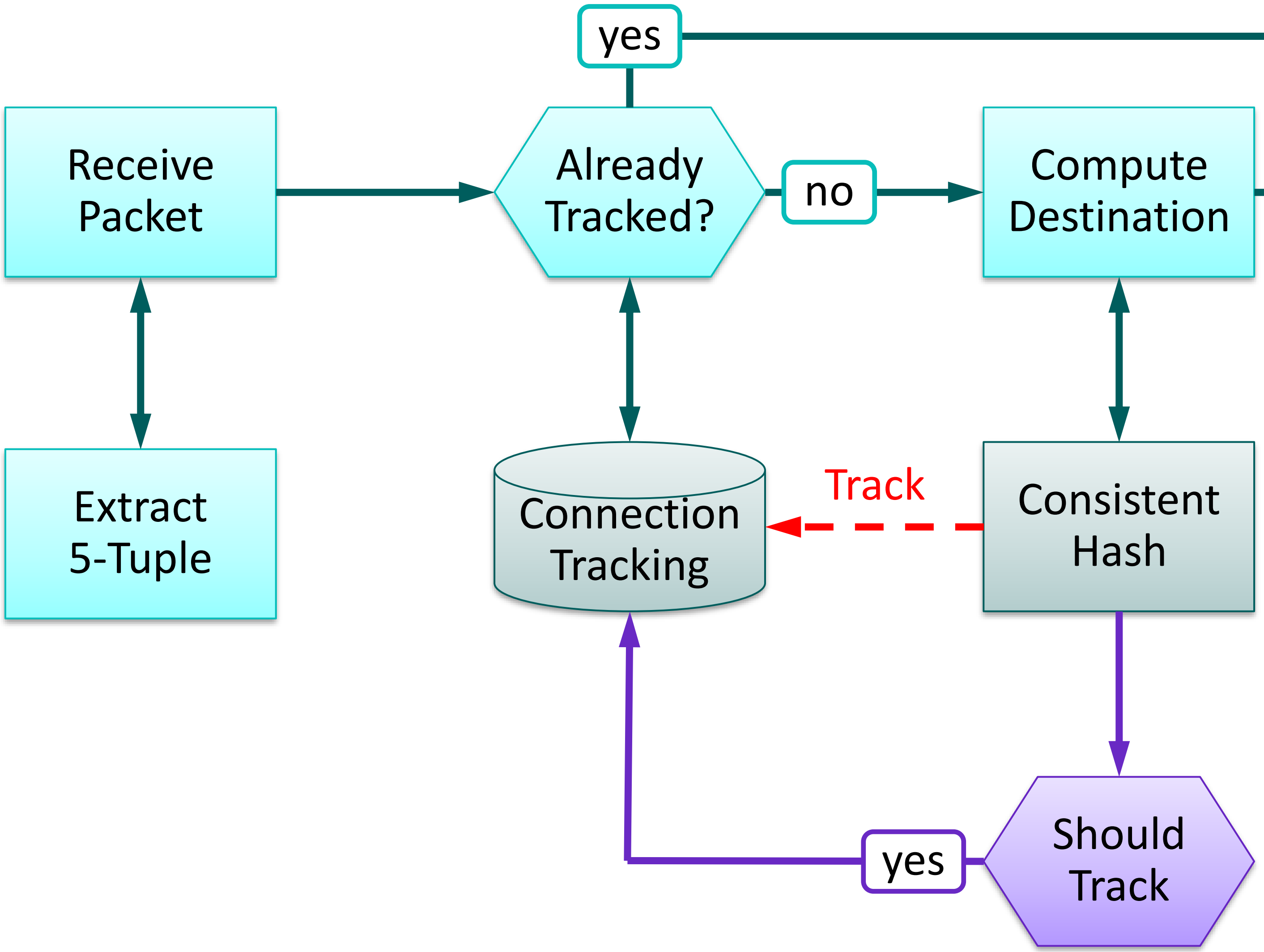
- *Horizon* set
  - Servers are added only from horizon set
- Warm-up period
  - Allow packet arrival from affected connections
  - Paced server addition → small horizon
    - E.g., if slower than TCP idle timeout then horizon can be a *single server*
- Removed servers are handled instantly
  - Transient failures are put in horizon set
    - Expected to be added back

Which Connections to Track?

- Answer:
  Ask the Consistent-Hash
- We implemented this for several consistent hash algorithms
  - Ring Hash
  - Highest Random Weight (HRW)
  - Table-based HRW
  - AnchorHash
- Very little overhead
  - Only 1 extra bit per entry in CH data structure
- See paper of details



JET Flow



Algorithm 1 JET

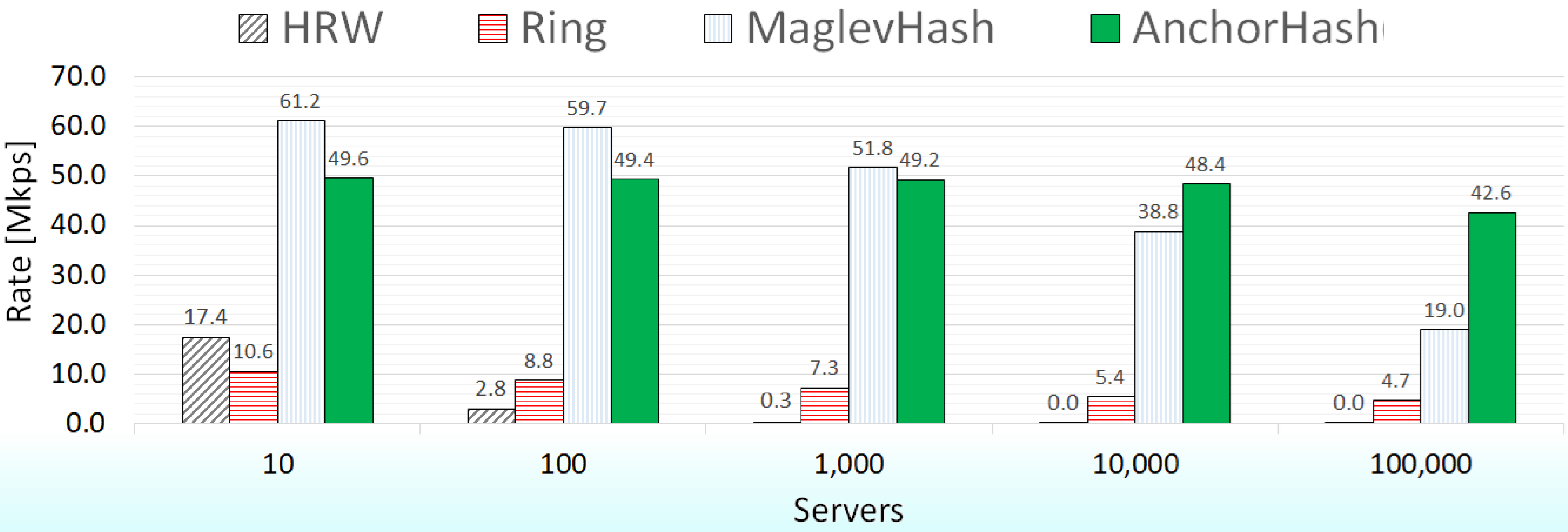
CT ▷ Connection tracking
CH ▷ Consistent hash

function GETDESTINATION(k)
  $s \leftarrow \mathrm{CT}[k]$
  if not s **then**
    $s \leftarrow \mathrm{CH}(\mathcal{W}, k)$
    if $s \neq \mathrm{CH}(\mathcal{W} \cup \mathcal{H}, k)$ **then** ▷ Should track?
      $\mathrm{CT}[k] \leftarrow s$
  return s

A Word on AnchorHash

– A new scalable consistent hash we developed
  - Ultra fast, small memory footprint, excellent balance
  - See our paper in ToN '21
  - Code available at https://github.com/anchorhash

– Works especially well with JET – no warmup period needed



Evaluation

– Event-based simulations
  - Inspired by evaluation of Cheetah, NSDI '20
    - » 468 servers
    - » Up to 40 backend changes per minute
    - » Varying connection rates

– Traces
  - Real traces
  - Synthetic traces

– Reproducibility
  - Code available at https://github.com/anchorhash/jetlb

PCC Violations

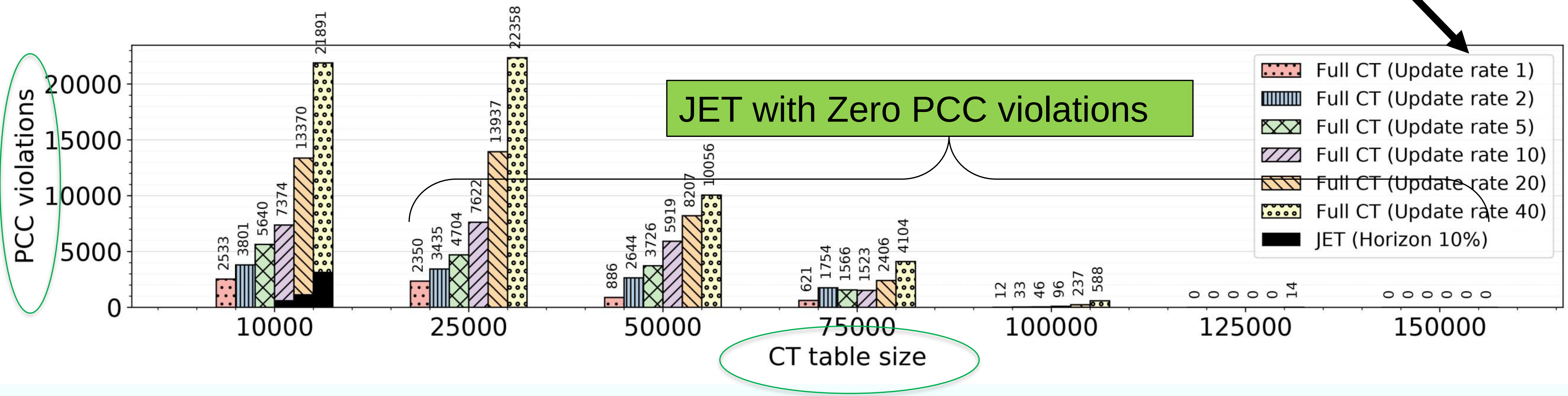
– 468 servers

– 100K active connections on average at any time

– 1K seconds (~16 minutes)

– JET (overlayed in black) with 10% horizon (47 servers)

Backend changes per minute



Balance, Tracking and Rate

- JET and full CT achieve the same balance
  - Use the same CH
- JET tracks less than 10% compared to full CT
- JET achieves higher rate due to smaller CT tables
  - Better caching

34.1M Packets
1.6M flows

	n=500				
	Table-based HRW		AnchorHash		MaglevHash
	Full CT	JET	Full CT	JET	Full CT
Maximum oversubscription	1.139 ±0.017	1.139 ±0.017	1.052 ±0.004	1.052 ±0.004	1.054 ±0.005
Tracked connections	1, 602, 007 ±0	145, 378 ±895.286	1, 602, 007 ±0	145, 543 ±230.205	1, 602, 007 ±0
Rate pkt/sec [millions]	22.883 ±2.573	45.567 ±4.113	22.702 ±0.134	30.856 ±0.187	23.446 ±2.839

Balance, Tracking volume and Rate

- JET and full CT achieve the same balance
  - Use the same CH
- JET tracks less than 10% compared to full CT
- JET achieves higher rate due to smaller CT tables
  - Better caching

34.1M Packets 1.6M flows	n=500				
	Table-based HRW		AnchorHash		MaglevHash
	Full CT	JET	Full CT	JET	Full CT
Maximum oversubscription	1.139 ±0.017	1.139 ±0.017	1.052 ±0.004	1.052 ±0.004	1.054 ±0.005
Tracked connections	1, 602, 007 ±0	145, 378 ±895.286	1, 602, 007 ±0	145, 543 ±230.205	1, 602, 007 ±0
Rate pkt/sec [millions]	22.883 ±2.573	45.567 ±4.113	22.702 ±0.134	30.856 ±0.187	23.446 ±2.839

Balance, Tracking volume and Rate

- JET and full CT achieve the same balance
  - Use the same CH
- JET tracks less than 10% compared to full CT
- JET achieves higher rate due to smaller CT tables
  - Better caching

34.1M Packets
1.6M flows

	n=500				
	Table-based HRW		AnchorHash		MaglevHash
	Full CT	JET	Full CT	JET	Full CT
Maximum oversubscription	1.139 ±0.017	1.139 ±0.017	1.052 ±0.004	1.052 ±0.004	1.054 ±0.005
Tracked connections	1, 602, 007 ±0	145, 378 ±895.286	1, 602, 007 ±0	145, 543 ±230.205	1, 602, 007 ±0
Rate pkt/sec [millions]	22.883 ±2.573	45.567 ±4.113	22.702 ±0.134	30.856 ±0.187	23.446 ±2.839

Balance, Tracking volume and Rate

- JET and full CT achieve the same balance
  - Use the same CH
- JET tracks less than 10% compared to full CT
- JET achieves higher rate due to smaller CT tables
  - Better caching

34.1M Packets
1.6M flows

	n=500				
	Table-based HRW		AnchorHash		MaglevHash
	Full CT	JET	Full CT	JET	Full CT
Maximum oversubscription	1.139 ±0.017	1.139 ±0.017	1.052 ±0.004	1.052 ±0.004	1.054 ±0.005
Tracked connections	1, 602, 007 ±0	145, 378 ±895.286	1, 602, 007 ±0	145, 543 ±230.205	1, 602, 007 ±0
Rate pkt/sec [millions]	22.883 ±2.573	45.567 ±4.113	22.702 ±0.134	30.856 ±0.187	23.446 ±2.839

More In The Paper

- JET formulation
- Pseudo-code for several consistent hash algorithms
- Theoretical guarantees
- Extensive evaluation
- Contact: galmen@stanford.edu

Thank you!

EXAMPLE

Adapting Ring hash to JET

Ring Hash 101

Ring: sorted list of tuples

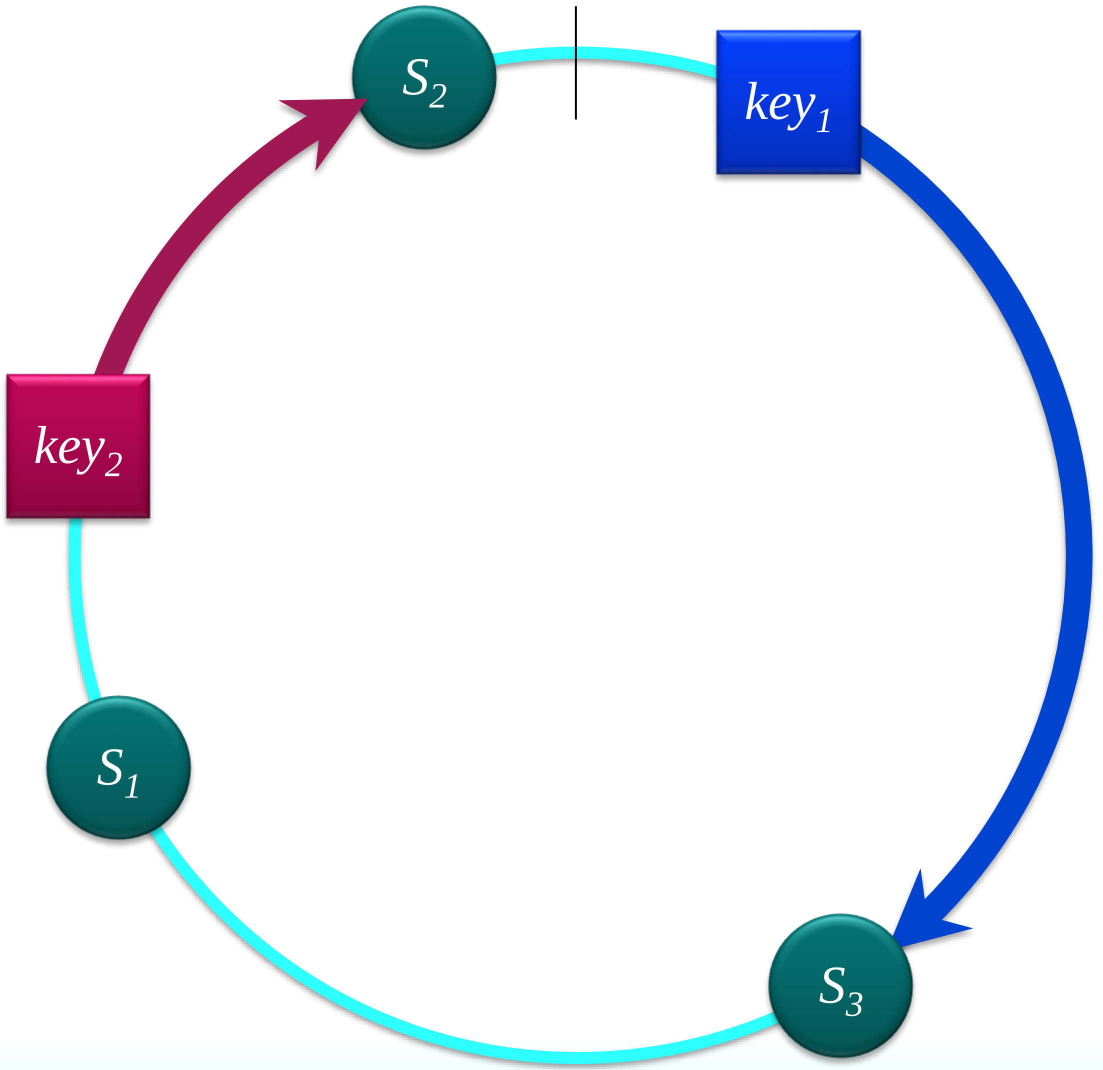
($hash(S_3)$, S_3)
($hash(S_1)$, S_1)
($hash(S_2)$, S_2)

Ring.get(*key*):

Search the sorted list for the successor of *hash(key)*

Example:

Ring.get(key_1) = S_3
Ring.get(key_2) = S_2



What if we add server H_1 ? (it is in the horizon set)

Ring: sorted list of tuples

($hash(H_1)$, H_1)

($hash(S_3)$, S_3)

($hash(S_1)$, S_1)

($hash(S_2)$, S_2)

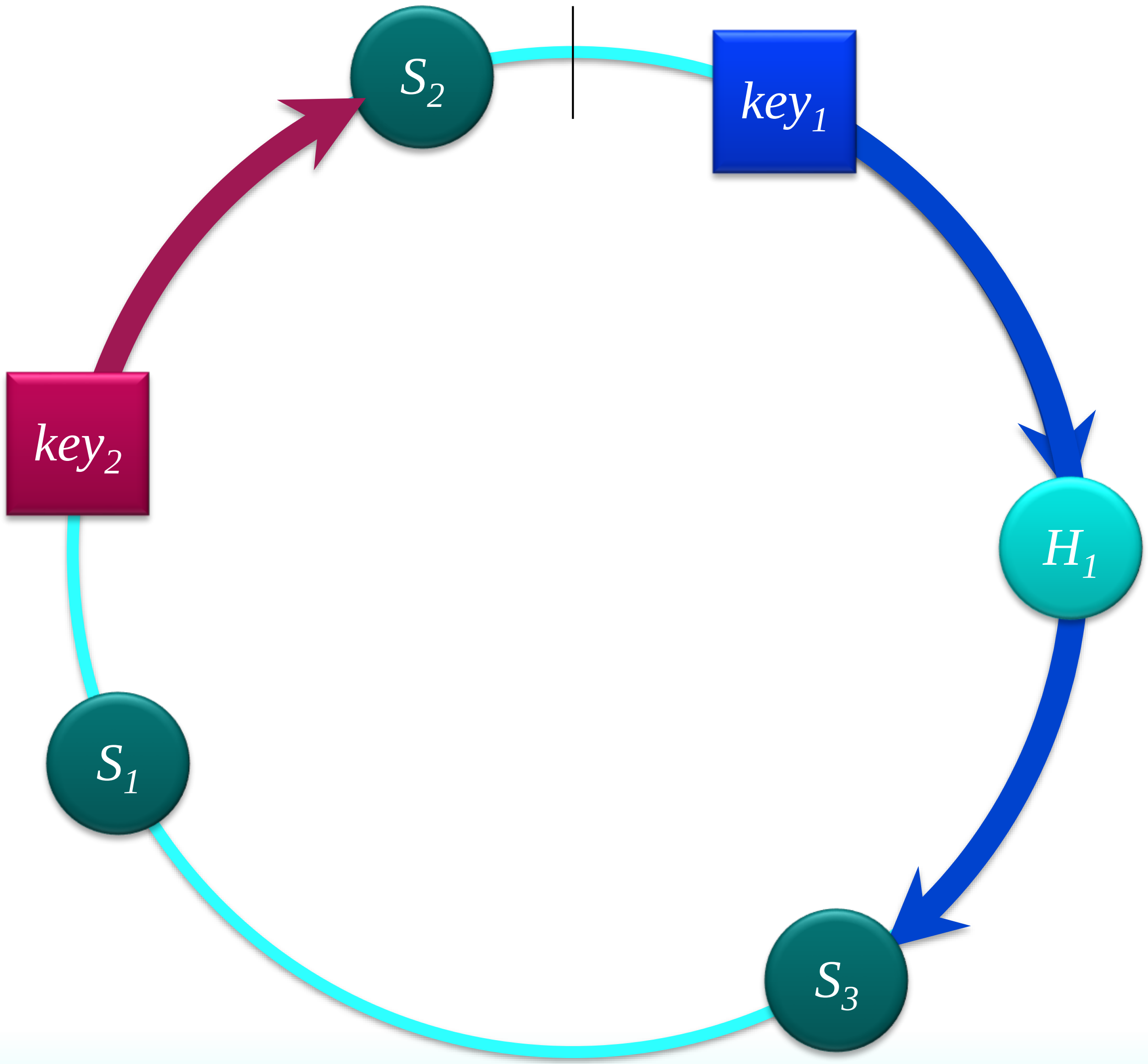
If we add server H_1 then:

Ring.**get**(key_1) = H_1 ← changed

→ key_1 should be tracked

Ring.**get**(key_2) = S_2 ← unchanged

→ key_2 should **not** be tracked



Add a "tracking" bit to each entry

Ring: sorted list of tuples

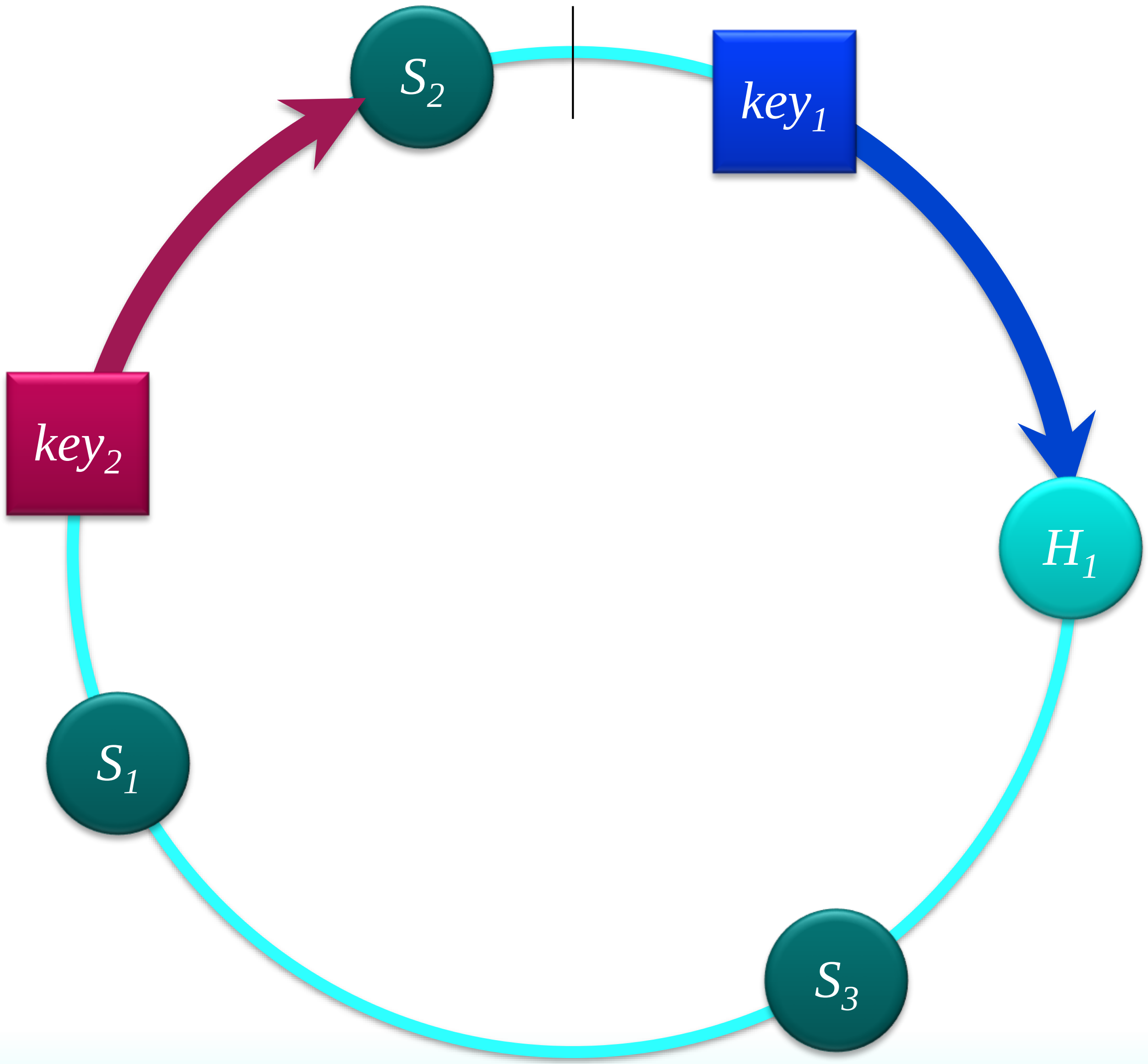
($hash(H_1)$, H_1, *Track=TRUE*)
($hash(S_3)$, S_3, *Track=FALSE*)
($hash(S_1)$, S_1, *Track=FALSE*)
($hash(S_2)$, S_2, *Track=FALSE*)

Ring.get(*key*):

Also return whether tracking is needed

Example:

Ring.get(key_1) = H_1, *Track=TRUE*)
Ring.get(key_2) = S_2, *Track=FALSE*)



Should still not return H_1

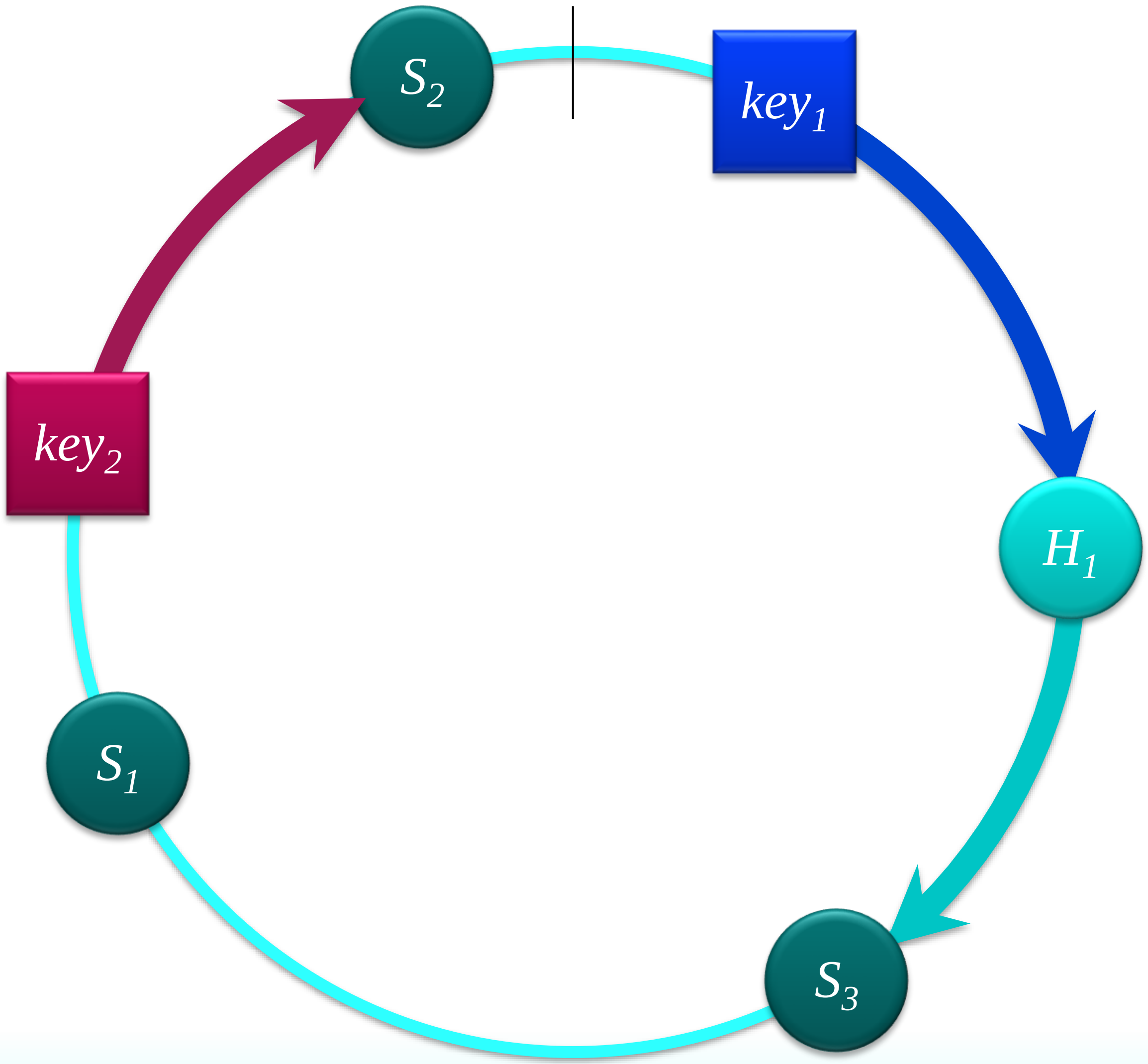
Ring: sorted list of tuples

($hash(H_1)$, S_3, $Track$=TRUE)

($hash(S_3)$, S_3, $Track$=FALSE)

($hash(S_1)$, S_1, $Track$=FALSE)

($hash(S_2)$, S_2, $Track$=FALSE)



Ring.get(key):

Return whether tracking is needed

Example:

Ring.get(key_1) = S_3, $Track$=TRUE)

Ring.get(key_2) = S_2, $Track$=FALSE)