A simple recipe for scaling Byzantine fault tolerant systems

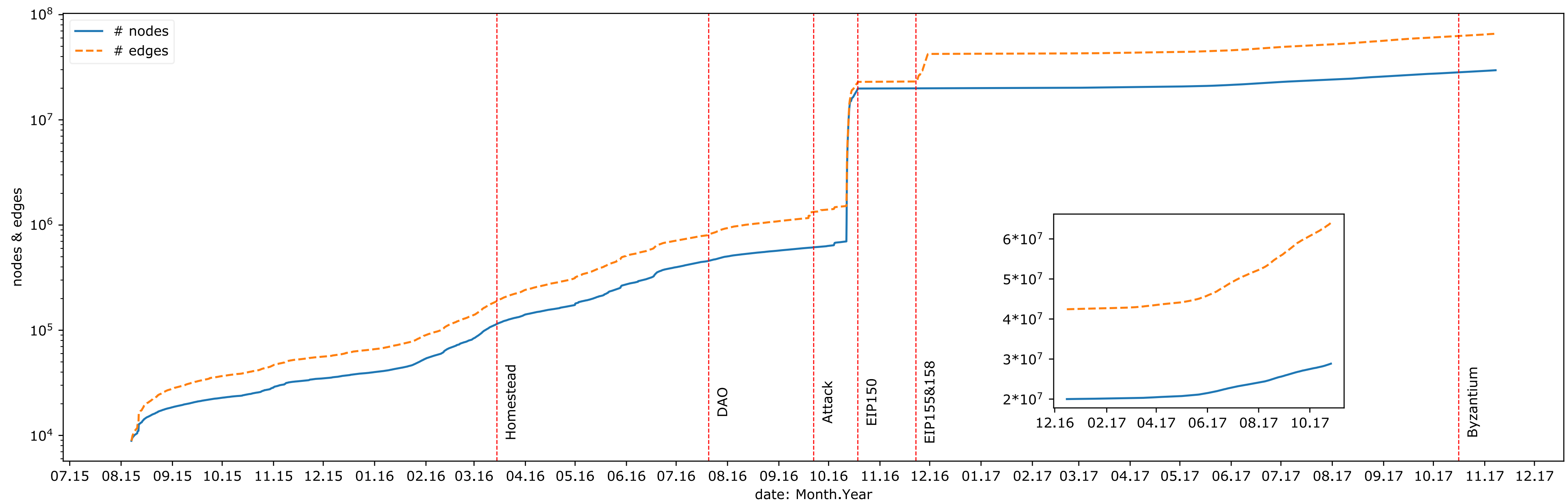
Fernando Pedone
Università della Svizzera italiana (USI)
Lugano, Switzerland

joint work with:
Alysson Bessani, University of Lisbon, Portugal
Tarcisio Ceolin Junior, Federal University of Santa Maria, Brazil
Paulo Coelho, Università della Svizzera italiana (USI), Switzerland
Fernando Dotti, Pontificia Universidade Católica do Rio Grande do Sul, Brazil
Enrique Fynn, Università della Svizzera italiana (USI), Switzerland
Christian Vuerich, Università della Svizzera italiana (USI), Switzerland

Context

- Byzantine fault tolerance is hot, again
  - ✦ Users have high expectations from systems
    - Scalability, availability, security, …
  - ✦ Some environments are unfriendly
    - Malicious participants
- Blockchain combines both

The rise of blockchain (Ethereum)

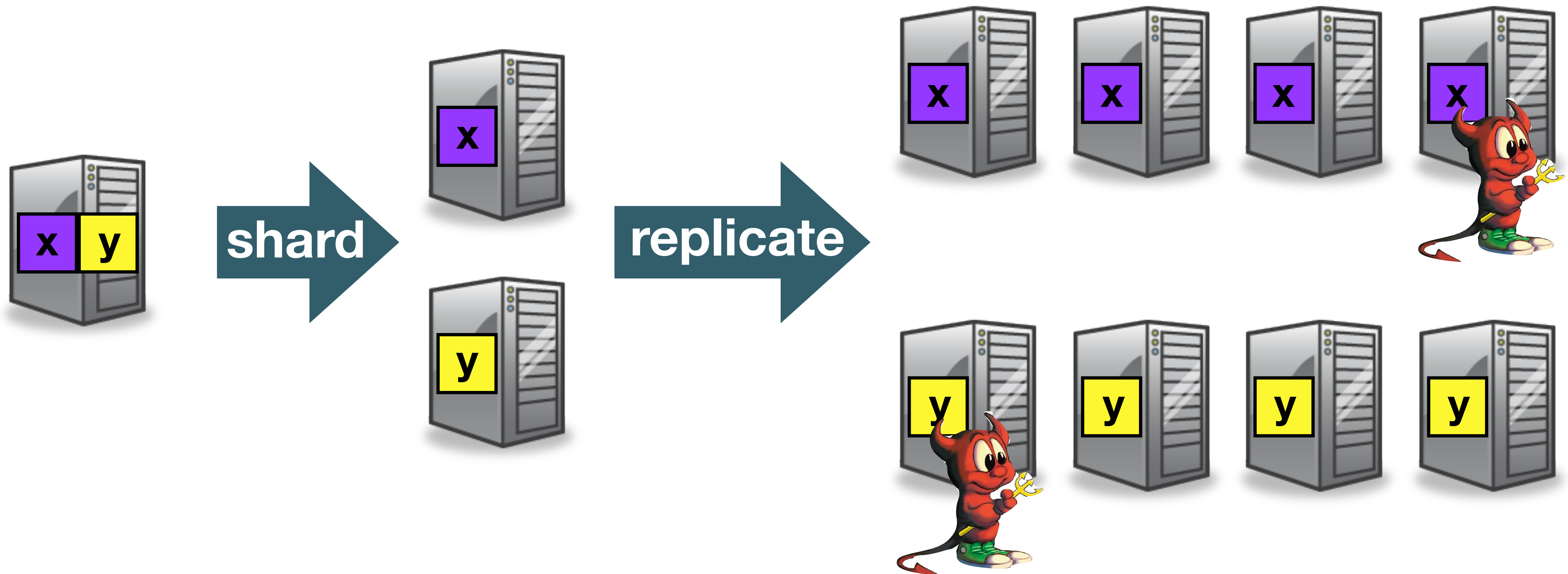


“Challenges and pitfalls of partitioning blockchains” with E. Fynn
Workshop on Byzantine Consensus and Resilient Blockchains (BCRB 2018)

Scaling blockchains

- This talk isn't about scaling blockchain
- But important aspect of scaling a system
- Possibly applicable to permissioned blockchains
- Not clear whether applicable to permissionless blockchains

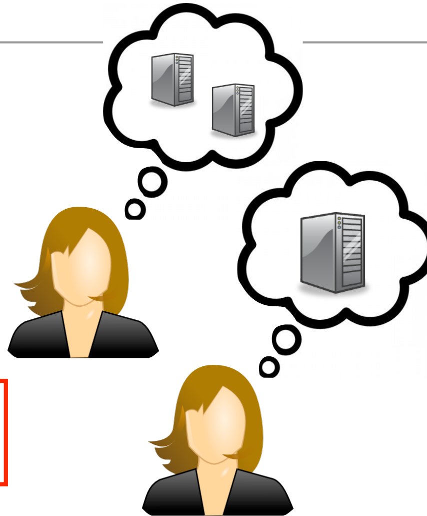
How to build scalable + available + robust systems?



Sharded & replicated systems

✦ Consistency criteria

- Weak consistency
- Strong consistency



✦ Handling commands in sharded+replicated systems

- Order commands
- Execute commands

Ordering commands with sharding+replication

- Intuitively
  - ✦ Any two replicas must order commands consistently within and across shards
- Formally
  - ✦ Define relation $<$ such that $m < m'$ iff there exists a non-faulty process that orders m before m'
  - ✦ Relation $<$ is acyclic

Encapsulating reliability and order

- BFT Atomic multicast abstraction
  - ✦ multicast(m, dst), where dst is one or more groups (shards)
  - ✦ deliver(m): event at a process after m has been ordered

Encapsulating reliability and order

- BFT Atomic multicast
  - ✦ Agreement: If a non-faulty process delivers message m, then eventually all non-faulty processes in m’s destination deliver m
  - ✦ Order: Relation < is acyclic
  - ✦ Validity: If a non-faulty process multicasts m, then eventually all non-faulty processes in m’s destination deliver m
  - ✦ Integrity: A non-faulty process delivers m at most once and only if some process multicast m

BFT Atomic multicast

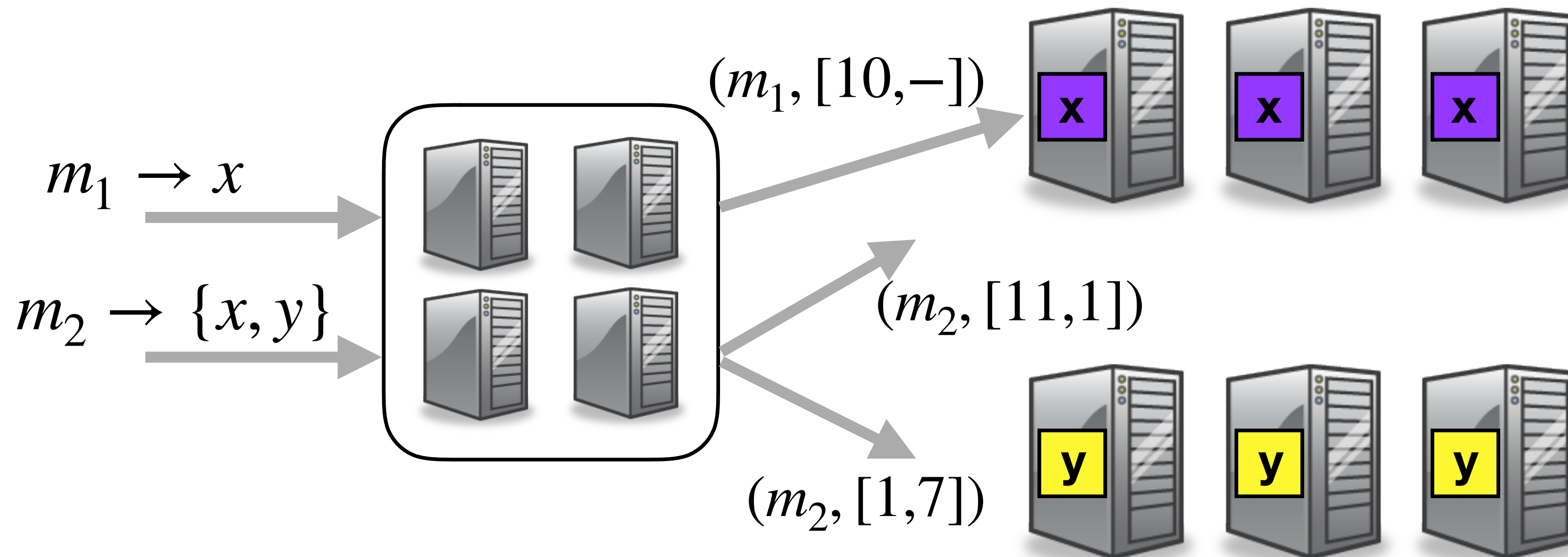
- Key requirements
  - ✦ Protocol must reuse existing BFT tools and libraries
  - ✦ Protocol must deliver scalable performance

Reusing existing systems

- Atomic broadcast
  - ✦ Special case of atomic multicast
  - ✦ Single group system
- Long history of contributions, including BFT
  - ✦ PBFT, BFT-SMaRt, Prime, HoneyBadgerBFT, …
  - ✦ Many academic contributions, not necessarily “usable” systems

Simple solution based on existing systems

- Naive BFT Atomic multicast
  - One group of processes orders all messages
  - Ordered messages relayed to destination groups



Naive BFT Atomic multicast isn’t good enough

- It doesn’t scale with number of groups
  - ✦ Ordering group eventually becomes performance bottleneck
- It is not suitable for geographically distributed settings
  - ✦ Latency induced by location of ordering group

Delivering scalable performance

- Genuine atomic multicast
  - ✦ Only sender and destinations should communicate to order a message
  - ✦ Performance can scale with the number of groups
  - ✦ Latency depended on message destinations only

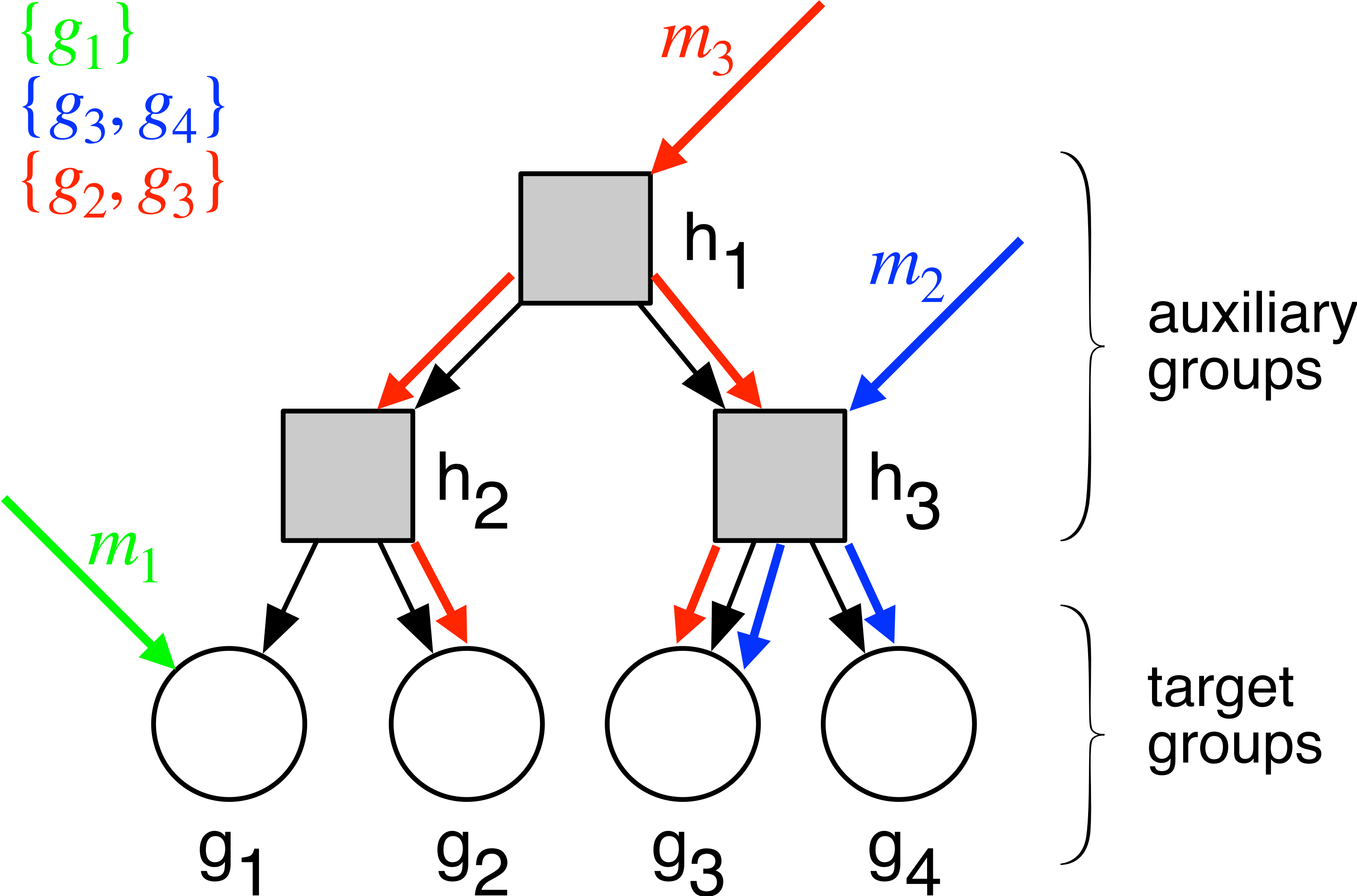
ByzCast: BFT Atomic multicast

- Compromise between reusability and scalability
  - ✦ Builds on BFT Atomic broadcast
    - actually, FIFO BFT Atomic multicast
  - ✦ Partially genuine
    - Genuine for single-group messages
    - Scales for single-group messages

ByzCast: BFT Atomic multicast

- Equip each group with a local FIFO BFT Atomic broadcast
- Create an overlay tree with all destinations
- To multicast message m
  - Order m first at the lowest common ancestor (LCA) of the message destination
  - Successively order m until destinations

ByzCast: BFT Atomic multicast



ByzCast: BFT Atomic multicast

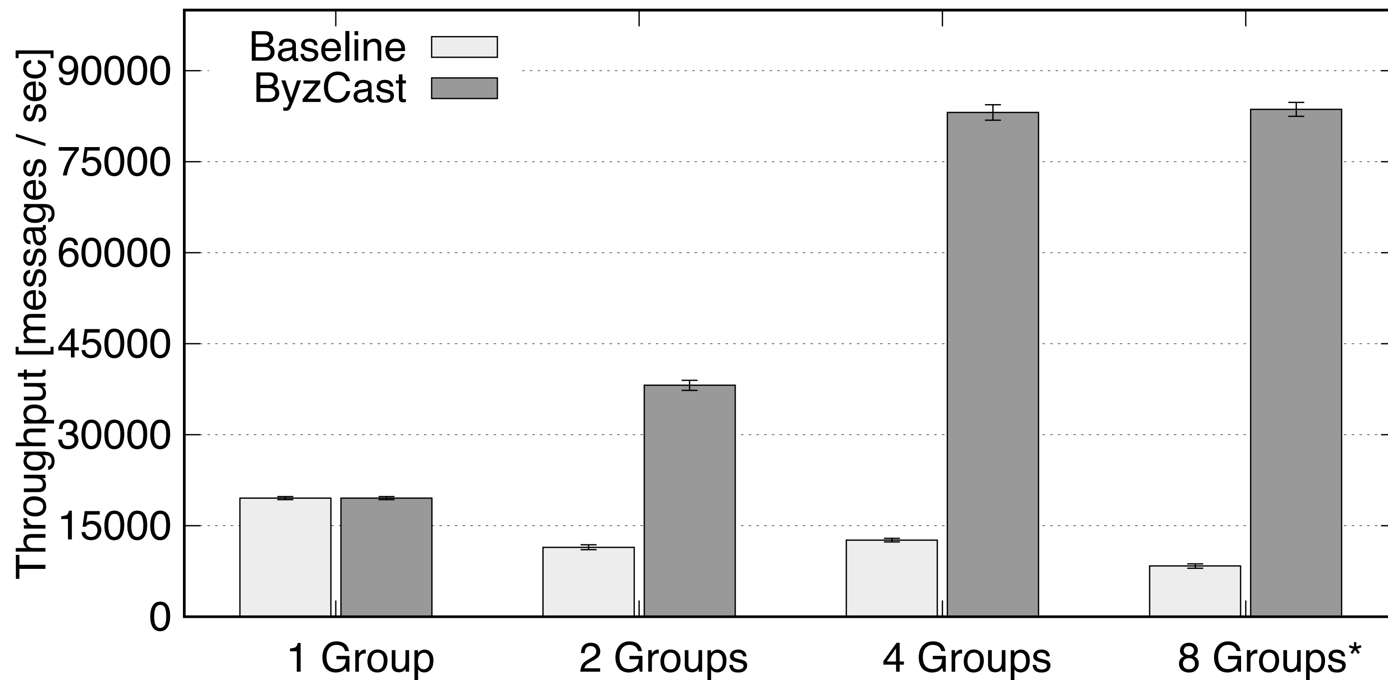
- Key invariants
  - Any two messages m and m′ atomically multicast to common destinations are ordered by at least one inner group x in the tree
  - If m is ordered before m′ in g, then m is ordered before m′ in any other group that orders both messages, thanks to the FIFO atomic broadcast used in each group

Performance evaluation

- Prototype (in Java) publicly available: https://github.com/tarcisiocjr/byzcast
- Each group uses BFT-SMaRt with 4 replicas
- 1 to 8 groups, 2-level overlay tree
- ByzCast vs. Naive multicast (Baseline)
  - Bottleneck in LAN
  - Latency in WAN

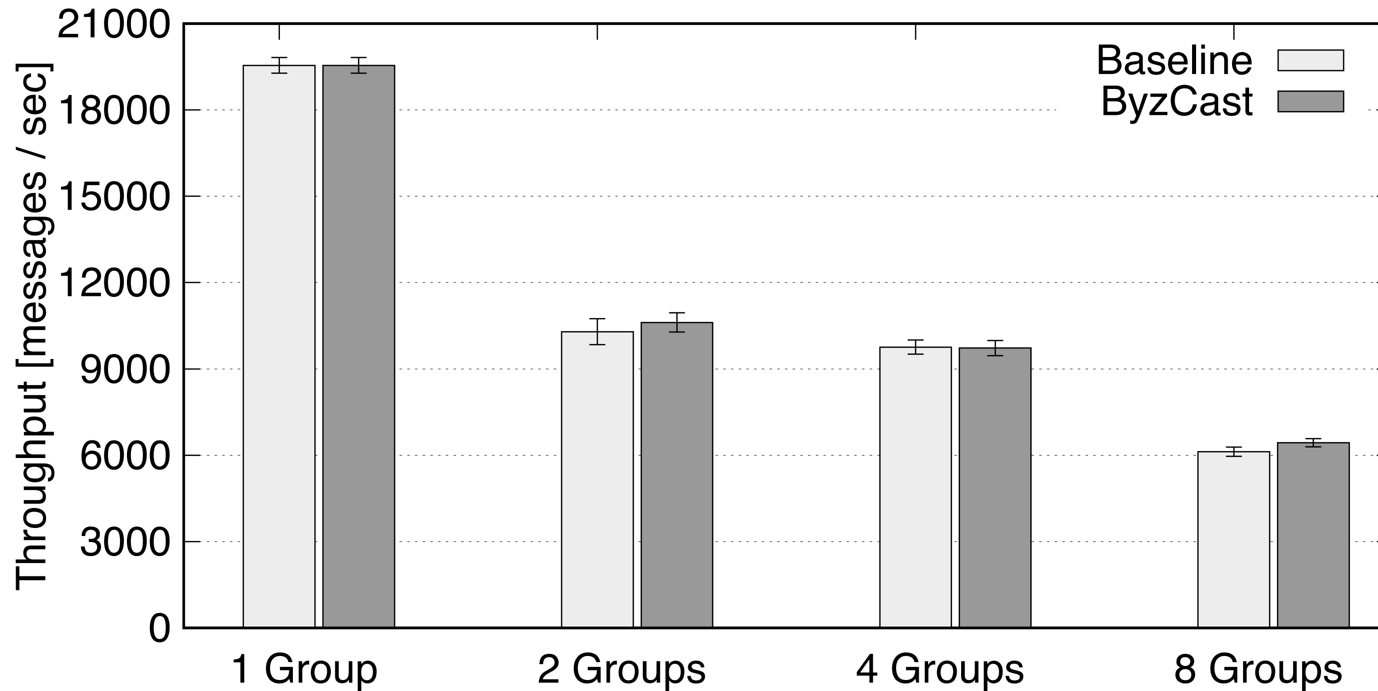
Performance in LAN

- Single-group messages: scalable performance



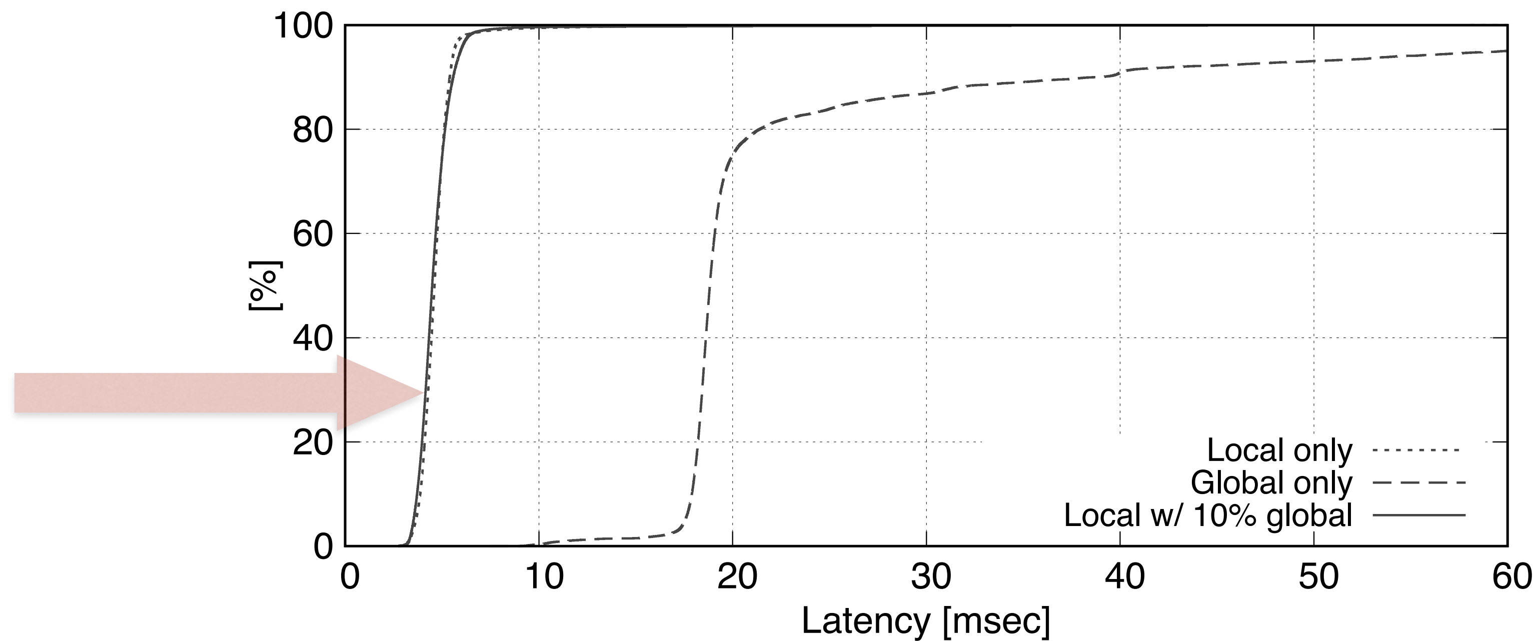
Performance in LAN

- Multi-group messages: ByzCast similar to Baseline



Performance in LAN

- Latency CDF: local msgs not delayed by global msgs



Performance in WAN

- Disaster-tolerant deployment
  - ✦ Failure of any single region
- Group members spread across 4 regions: California (CA), Frankfurt (EU), North Virginia (VA) and Japan (JP)

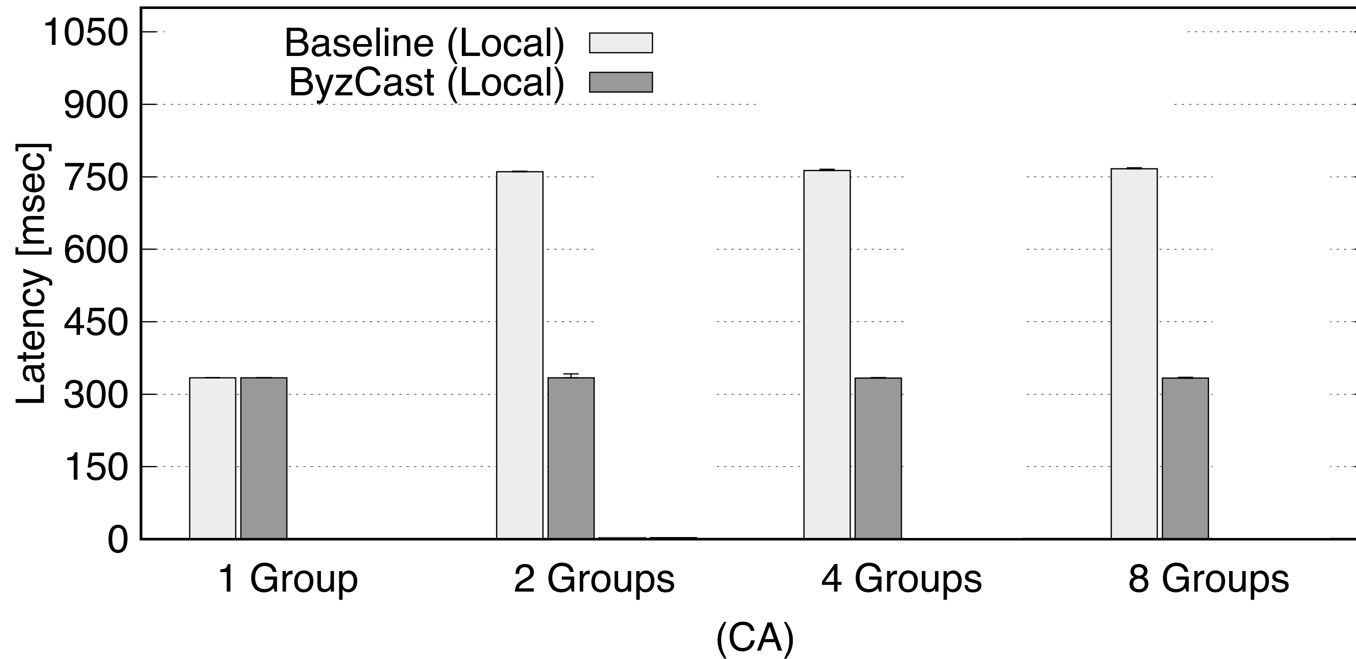
	EU	CA	VA	JP
CA	165	–	70	112
VA	88	70	–	175
JP	239	112	175	–

Latencies between regions in milliseconds.

Throughput [messages /
15000
12000
9000
6000
3000
0
100
80
60
[%]
40
20
0
100
80
60
[%]
40

Performance in WAN

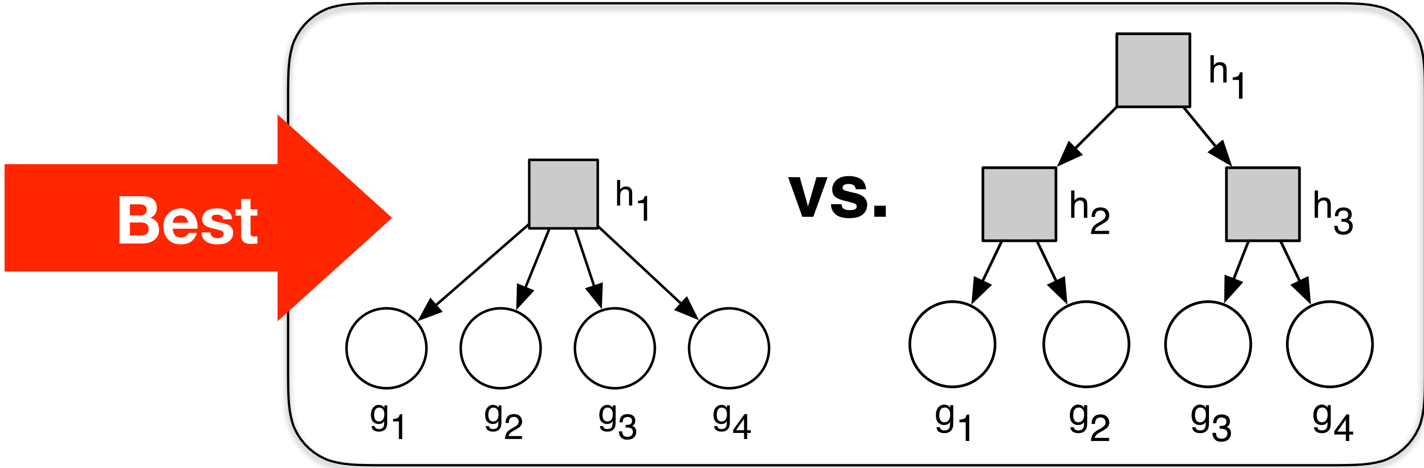
- Latency: ByzCast more efficient for single-group msgs



Building the overlay tree

- Short trees are better (for latency), but…
- Inner nodes may hamper performance

Workload	Destinations	Throughput/ destination	2-level h1	3-level h1	h2	h3
Uniform	any 2 groups out of 4	~1200 msgs/sec	7.2k	4.8k	6k	6k

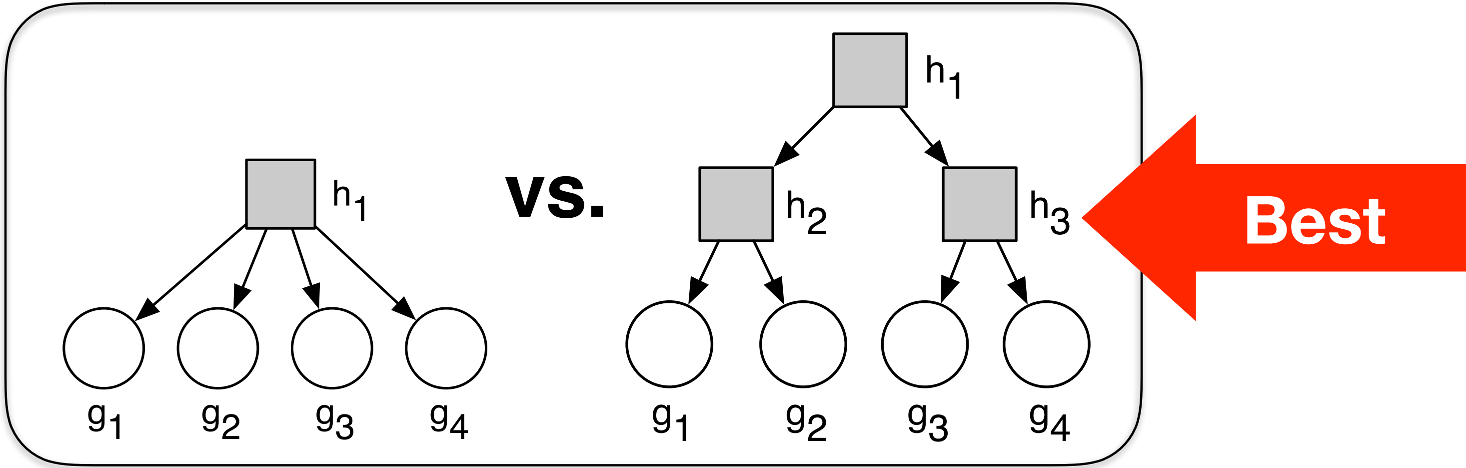


Max throughput per group = 10k msgs/sec

Building the overlay tree

- Short trees are better (for latency), but…
- Inner nodes may hamper performance

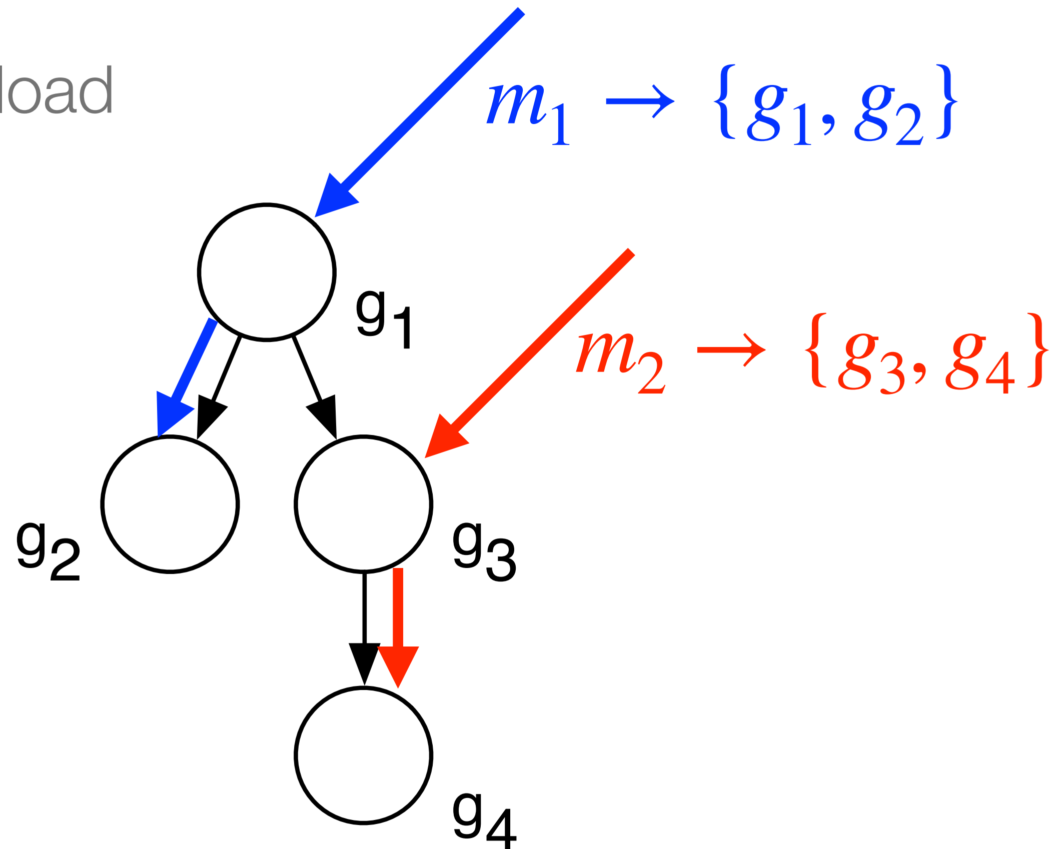
Workload	Destinations	Throughput/ destination	2-level	3-level		
			h1	h1	h2	h3
Skewed	{g1,g2},{g3,g4}	~9000 msgs/sec	18k	0k	9k	9k



Max throughput per group = 10k msgs/sec

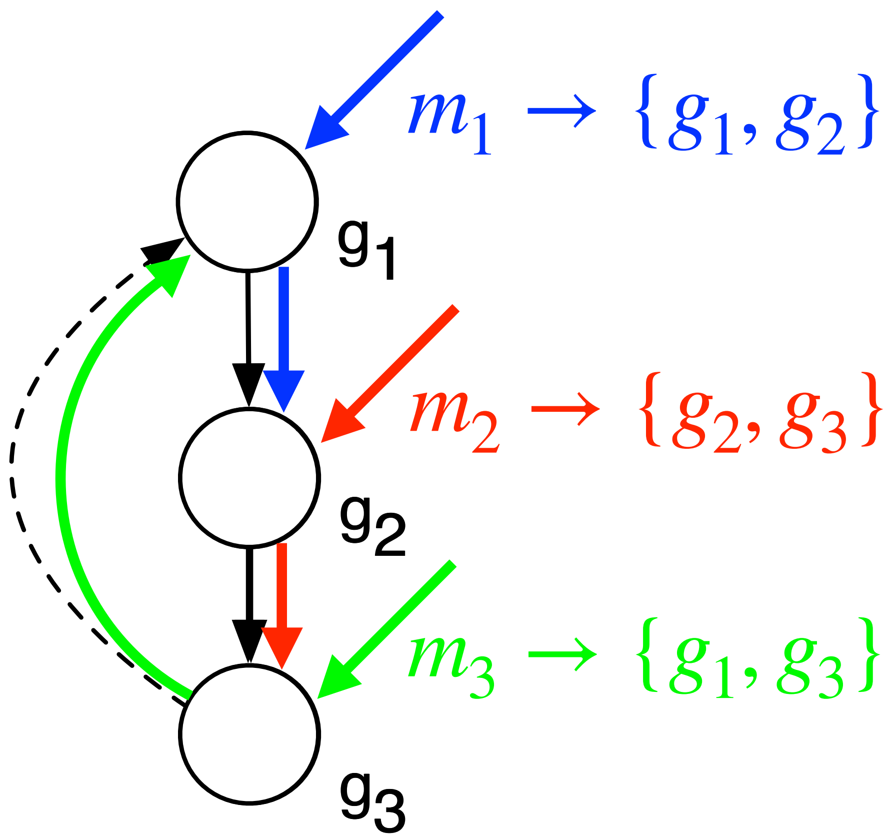
Building the overlay tree

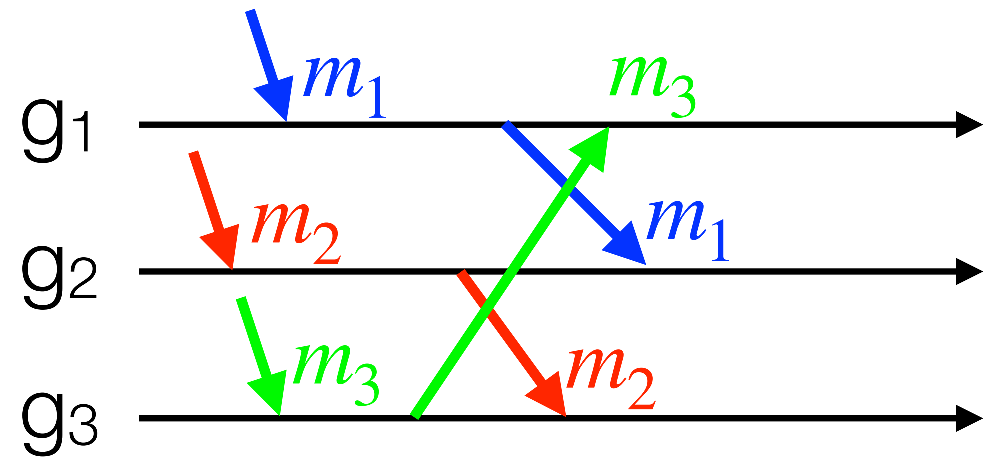
- With no auxiliary groups genuine protocol possible sometimes, but not always
  - e.g., skewed workload



Why a tree?

- Case 1: Cycles





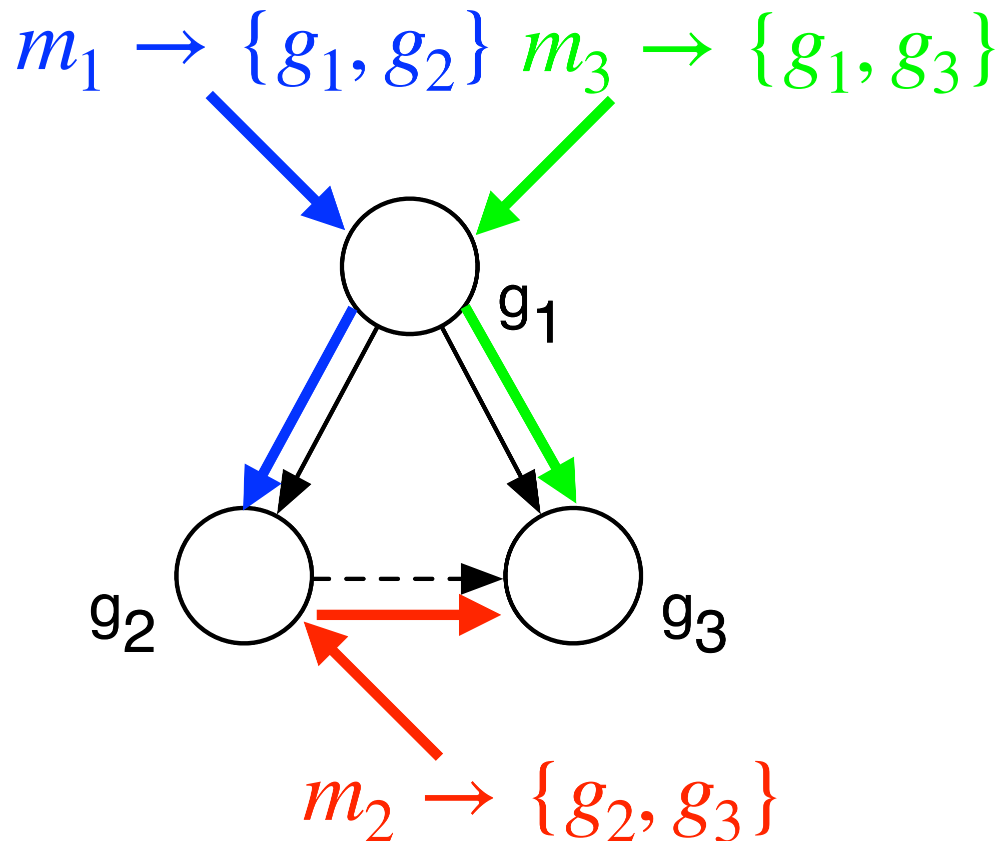
from g_3

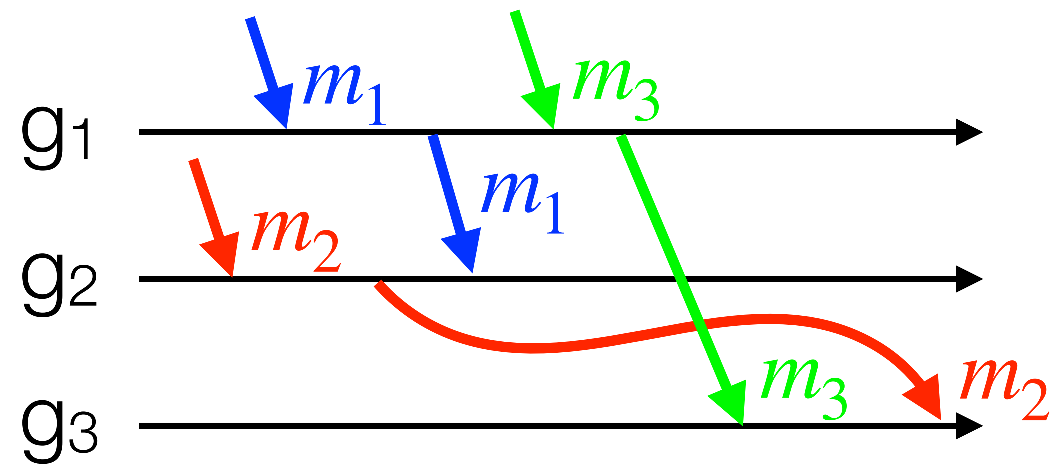
$$m_1 < m_3 < m_2 < m_1$$

from g_1 from g_2

Why a tree?

- Case 2: Nodes with two (or more) incoming arrows





from g_1

$$m_2 < m_1 < m_3 < m_2$$

from g_2 from g_3

Final remarks

- More details in DSN 2018 paper
- Building tree overlay is a hard problem
- Workload-based dynamic tree configuration
- Application in blockchain, mostly permissioned
- How malicious attacks propagate in BFT protocol composition? — thank you Yair! :-)