Hein Meling

Department of Electrical Engineering and Computer Science

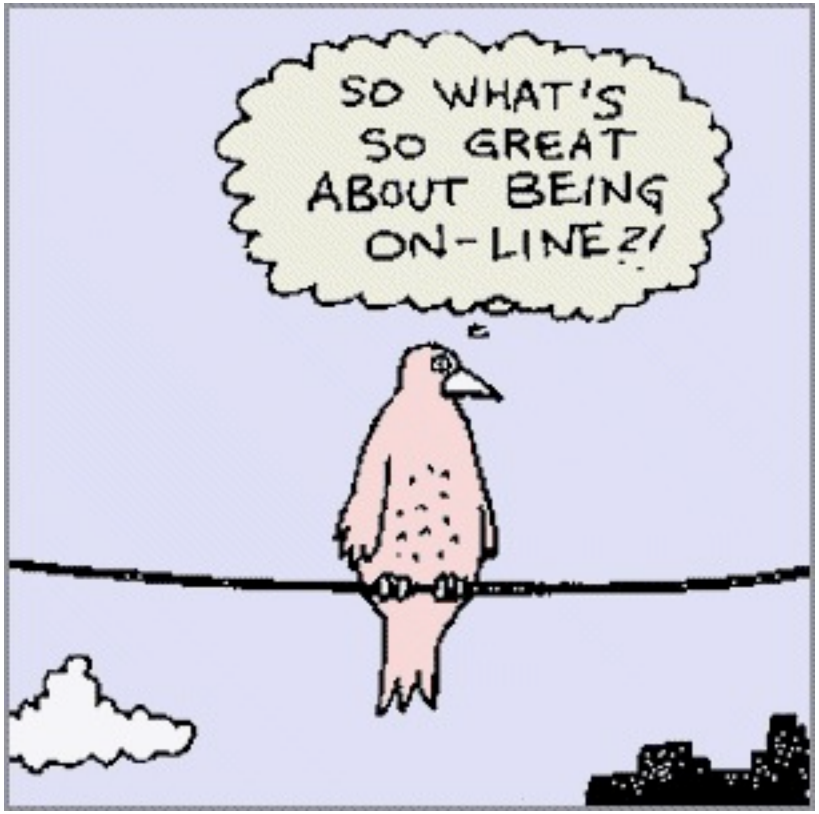
University of Stavanger, Norway

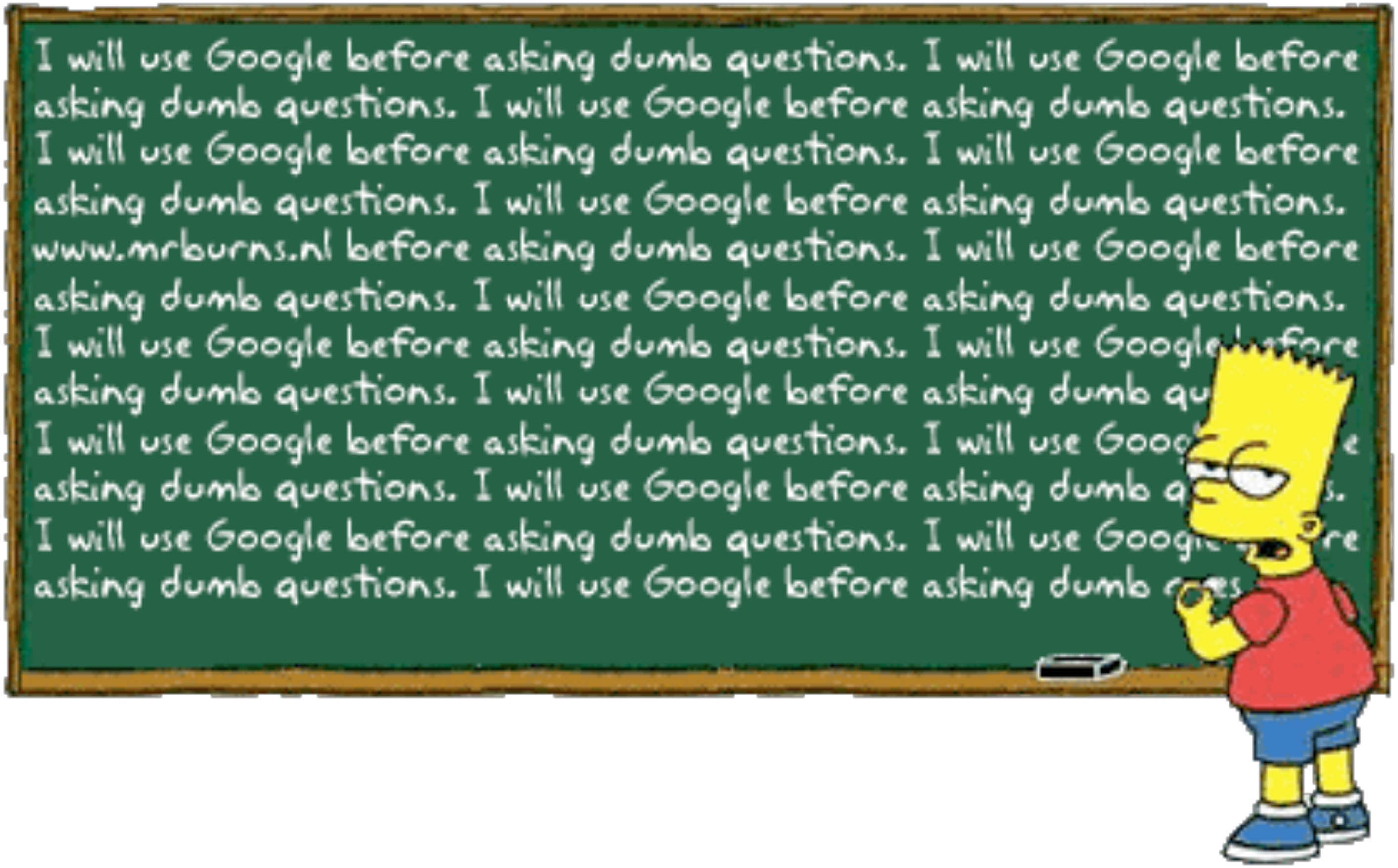
Self-repairing Replicated Systems and Dependability Evaluation

Toronto, August 27, 2010

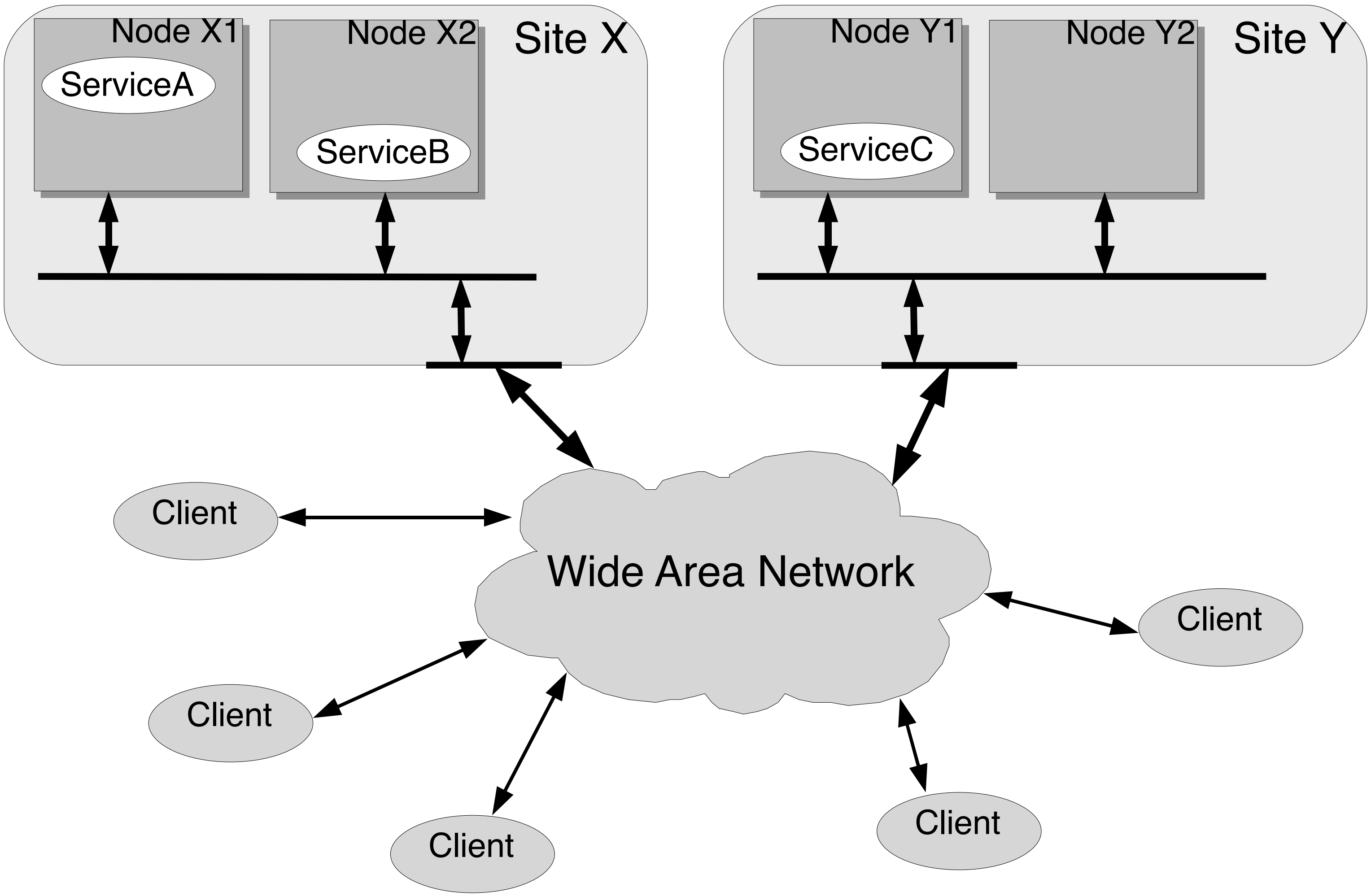
CANOE Workshop

University of Stavanger

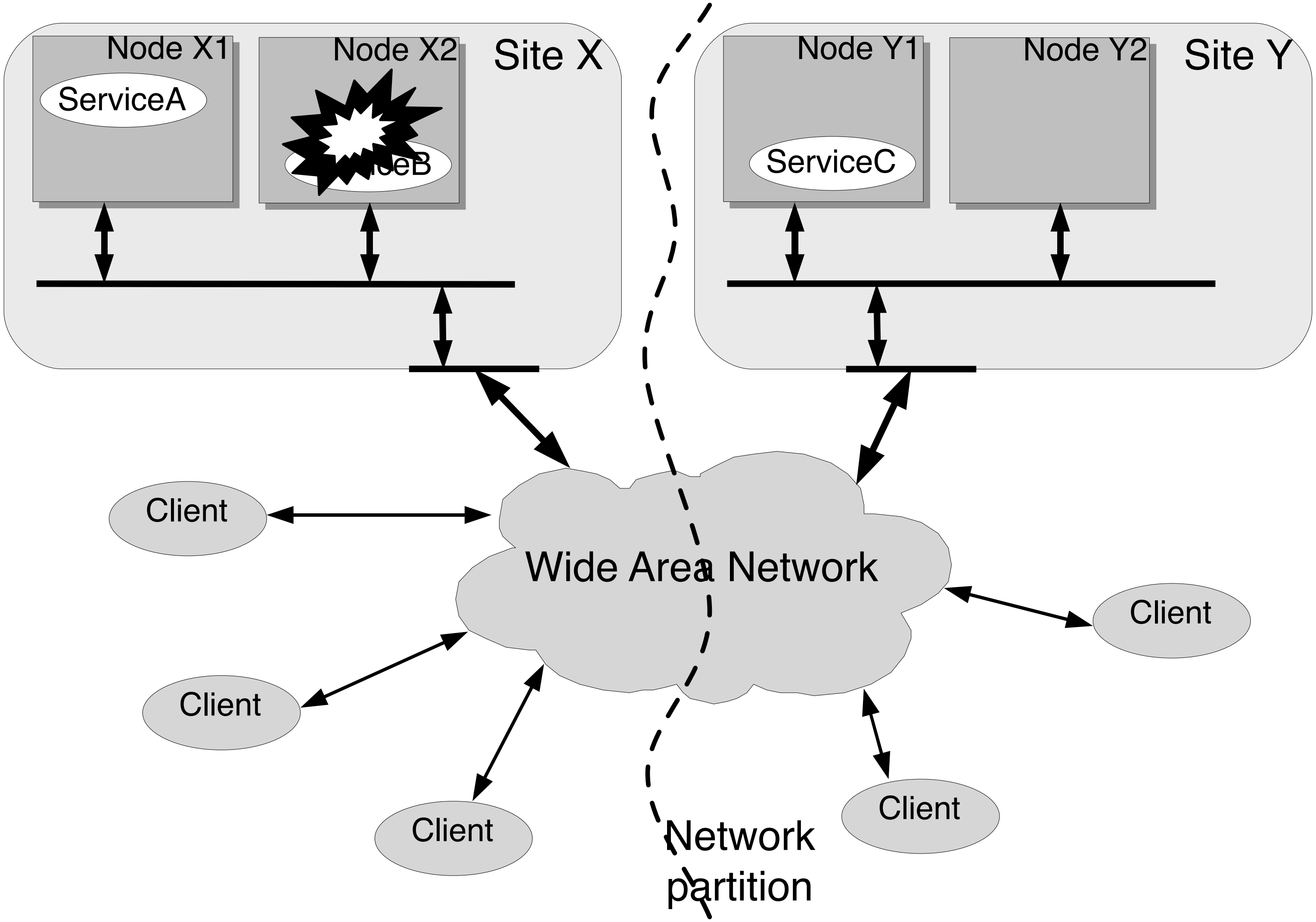

SO WHAT'S SO GREAT ABOUT BEING ON-LINE?!


I will use Google before asking dumb questions. I will use Google before
asking dumb questions. I will use Google before asking dumb questions.
I will use Google before asking dumb questions. I will use Google before
asking dumb questions. I will use Google before asking dumb questions.
www.mrburns.nl before asking dumb questions. I will use Google before
asking dumb questions. I will use Google before asking dumb questions.
I will use Google before asking dumb questions. I will use
asking dumb questions. I will use Google before asking dumb
I will use Google before asking dumb questions. I will use
asking dumb questions. I will use Google before asking dumb
I will use Google before asking dumb questions. I will use
asking dumb questions. I will use Google before asking dumb

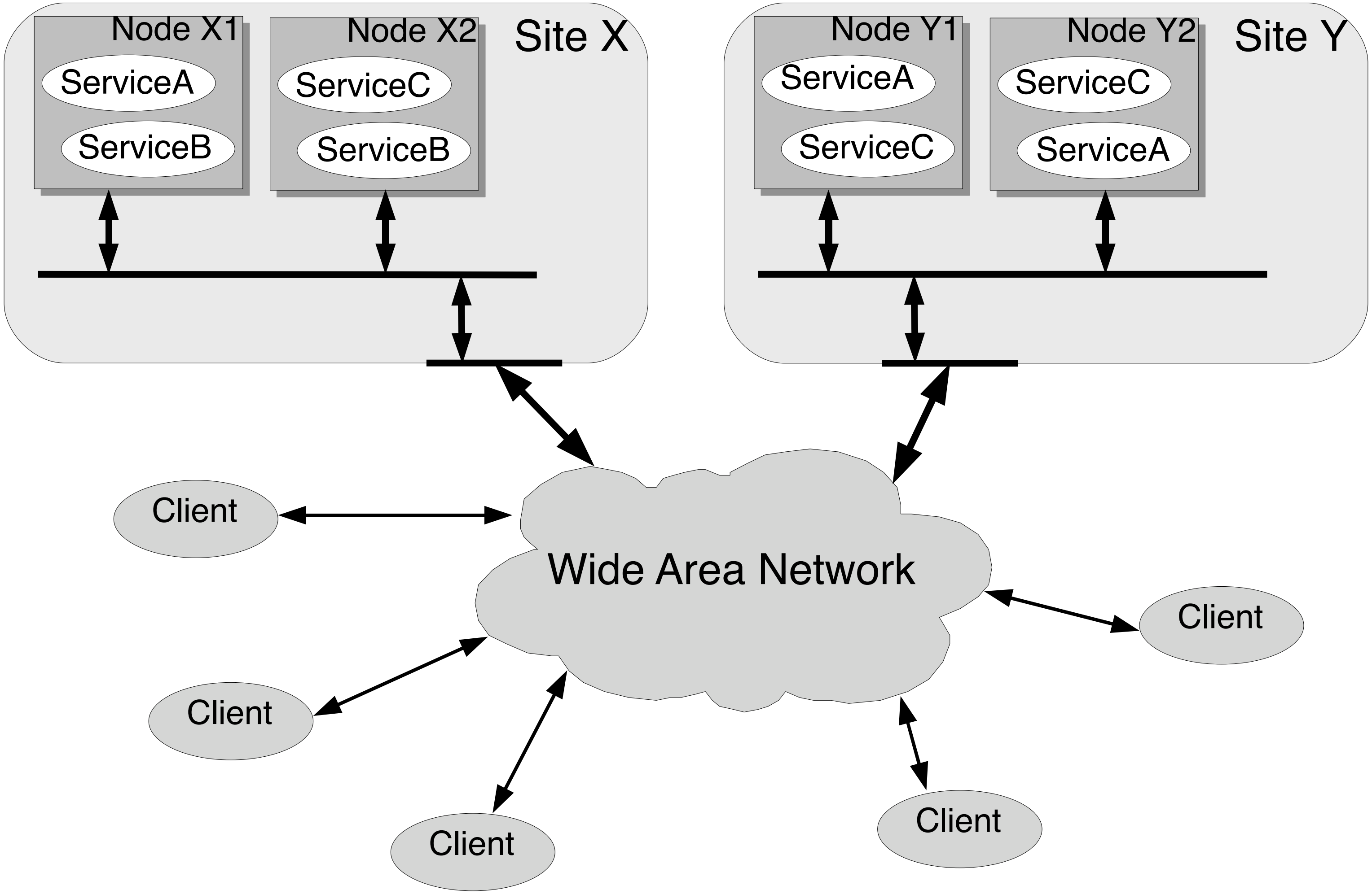
Context – Multiple Data Centers



Context – Failures will occur



Common Solution is Redundancy

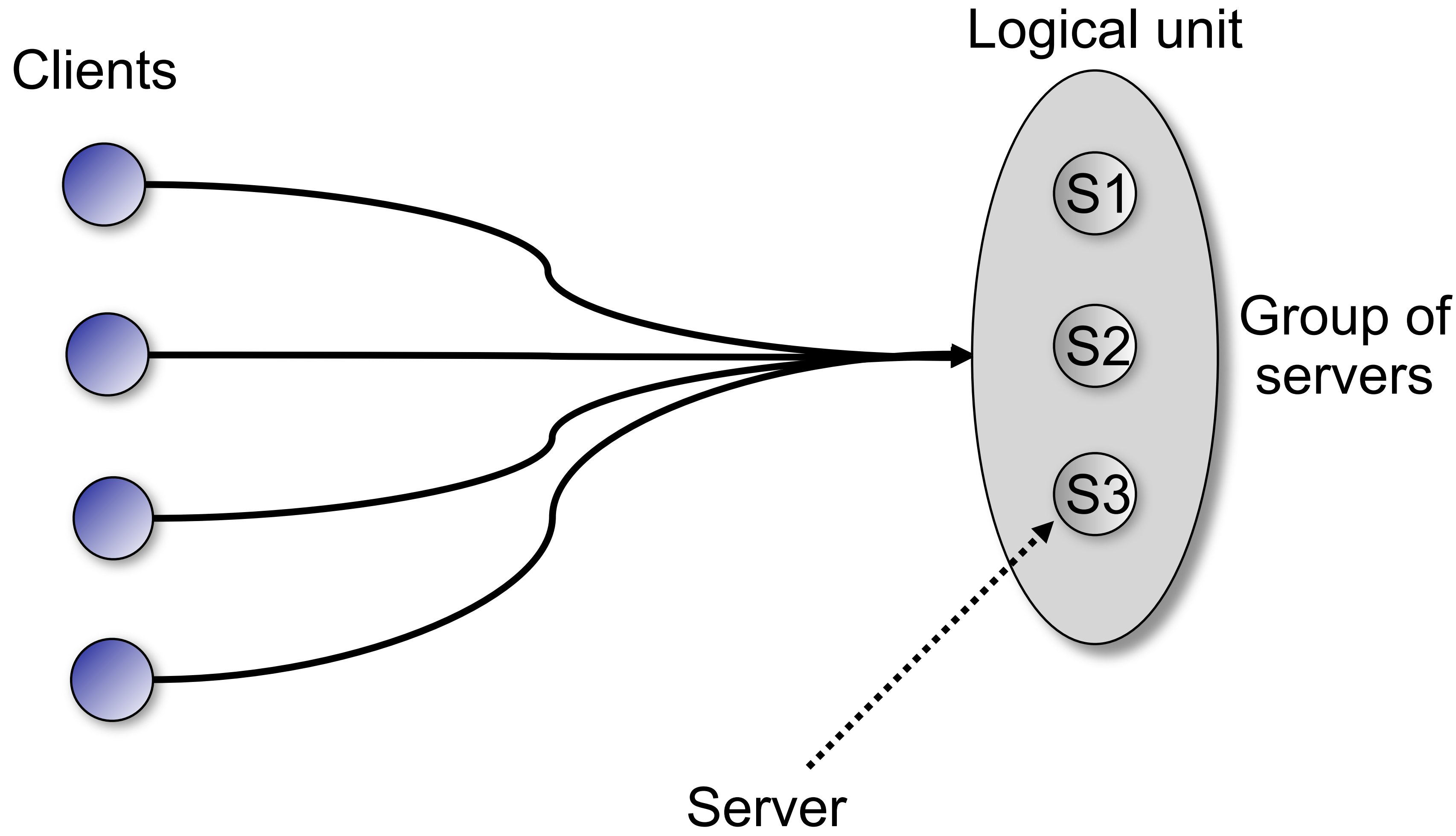


Middleware for Fault Tolerance

University of Stavanger

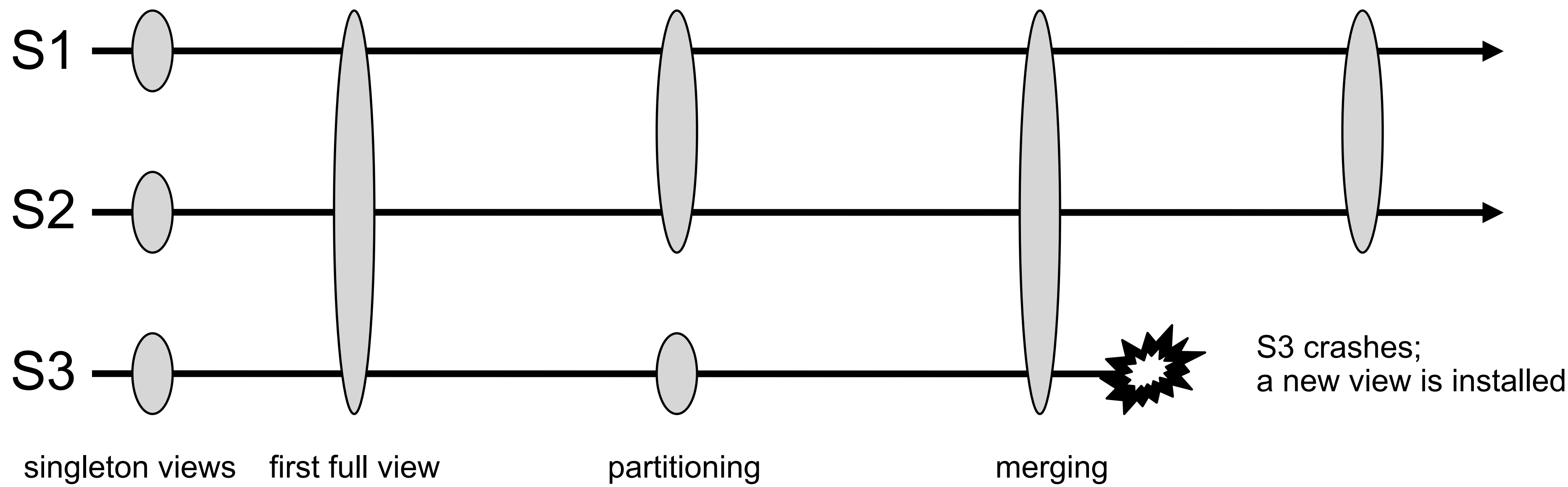
- It is difficult to support fault tolerance
  - Tolerate object, node and *network failures*
- Techniques
  - Redundancy
  - Masking failures (failover)
- Reuse fault tolerance mechanisms
  - Use a group communication system (e.g. Jgroup or Spread)
- Focus on development issues

Group Communication



The Group Membership Service

University of Stavanger



Middleware for Fault Treatment

University of Stavanger

- Further improve the system's dependability characteristics
  - Consider: Deployment and operational aspects
- Autonomous Fault Treatment
  - Recovery from node, object and *network failures*
  - Not just tolerate faults, repair them as well
  - Without human intervention
  - *Let groups be self-healing* (deal with its own internal failures)
- Goal: Minimize the time spent in a state of reduced failure resilience

Evaluation Techniques

- Trivial performance evaluation of repair mechanism
  - For a single failure injection
- But more interesting
  - Can we find a way to quantify/predict the improvement in availability by running experiments?
  - (Without running them for many years to get the exact numbers.)

Moving to large-scale (Cloud)

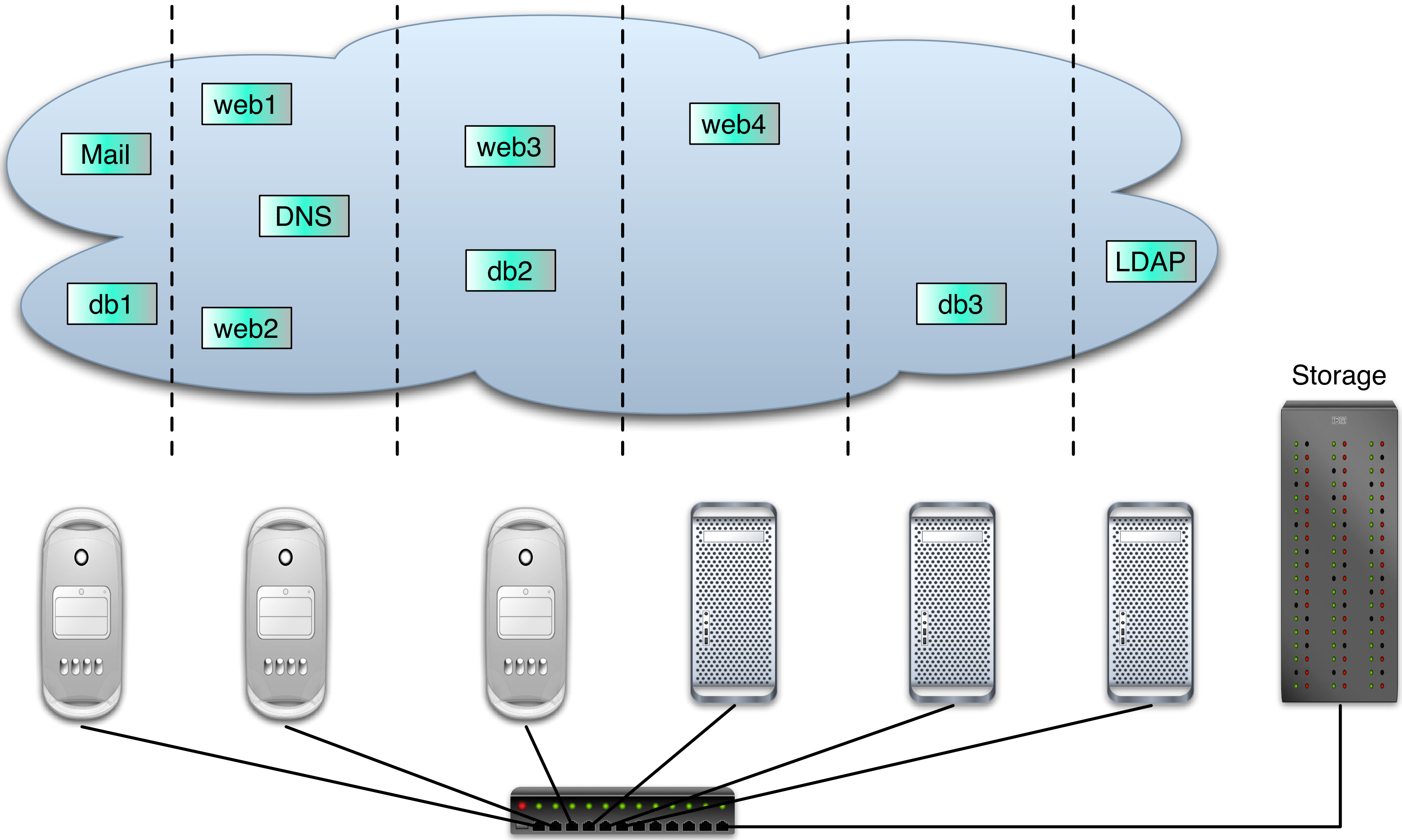
University of Stavanger

- Assume now the number of services to deploy becomes very large
  - We need to find placements for the services to avoid bottlenecks
  - Multiple conflicting requirements/goals for these services
  - Placement is a multi-criteria optimization problem
- Placement becomes NP-hard
  - Centralized optimization techniques fall short quickly
- Also, if it were possible to compute the optimal placement
  - Would it still be valid when we are ready to deploy/reconfigure?
- Distributed heuristic to compute near optimal placements
  - Based on a technique called Cross-Entropy Ant System

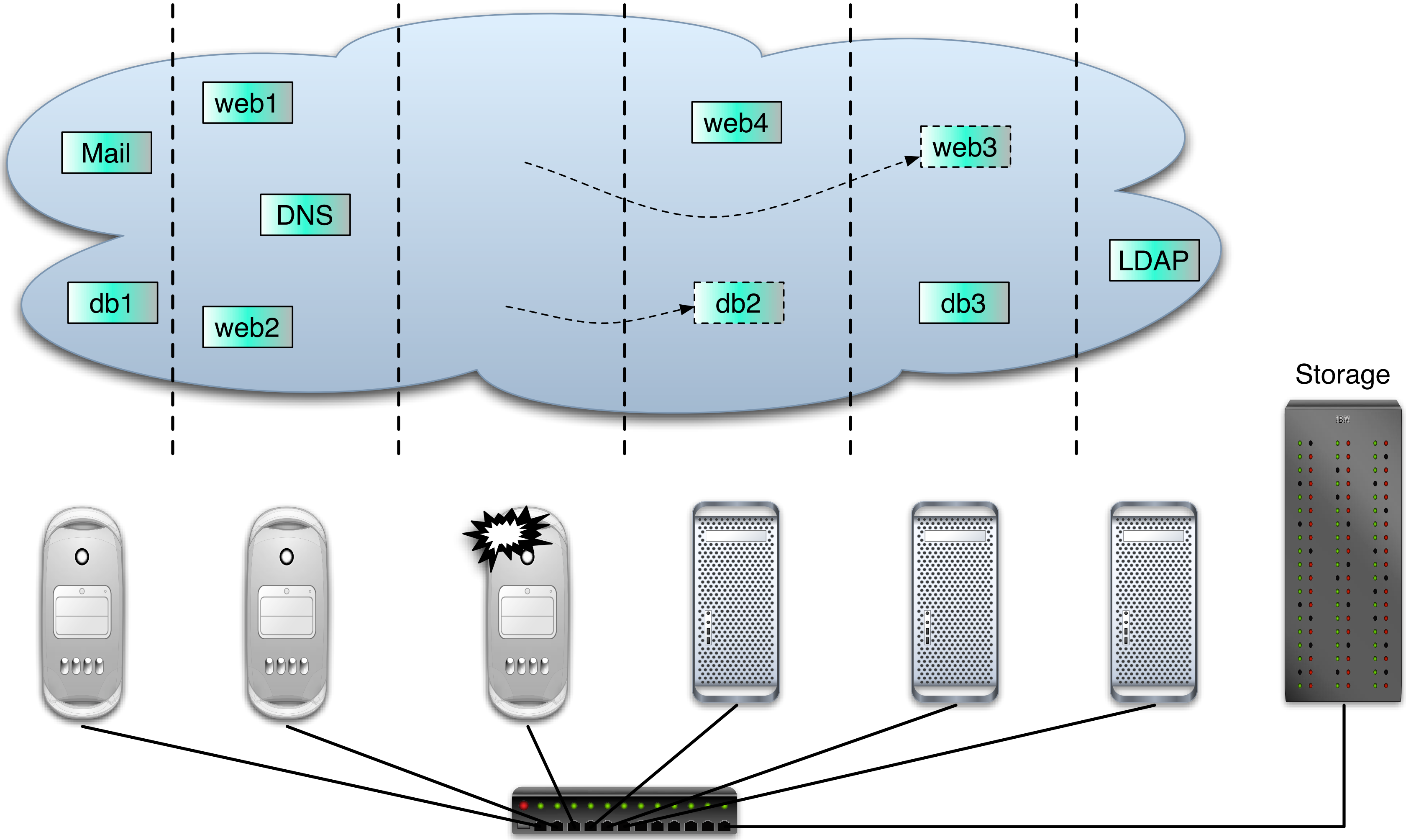
Outline

- Introduction and motivation
- Related work
- Distributed Autonomous Replication Management (DARM)
- Simple Network Partition Evaluation of DARM
- Dependability Evaluation Technique
- Concluding remarks

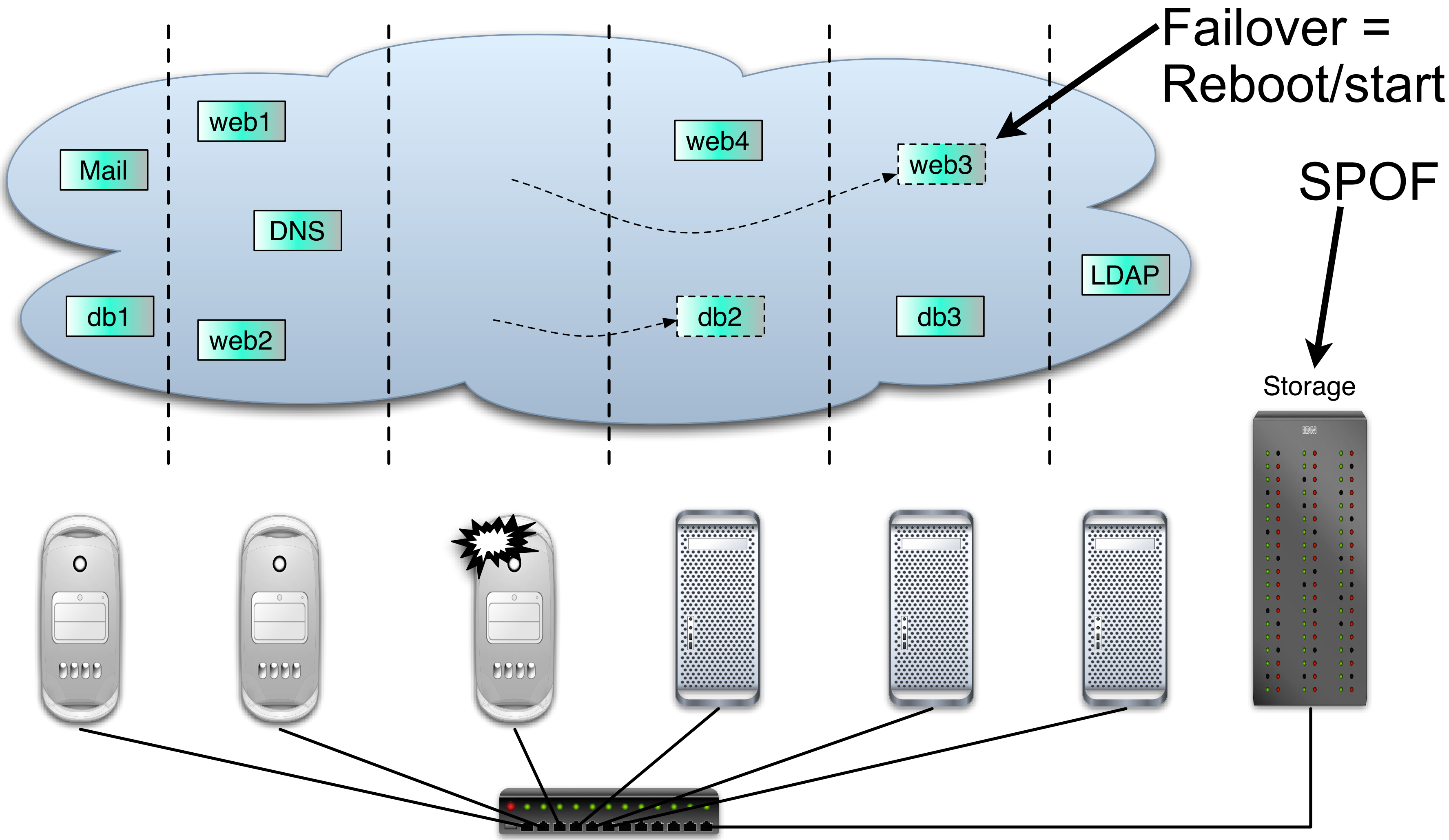
Related work: Virtualization



Related work: Virtualization



Related work: Virtualization



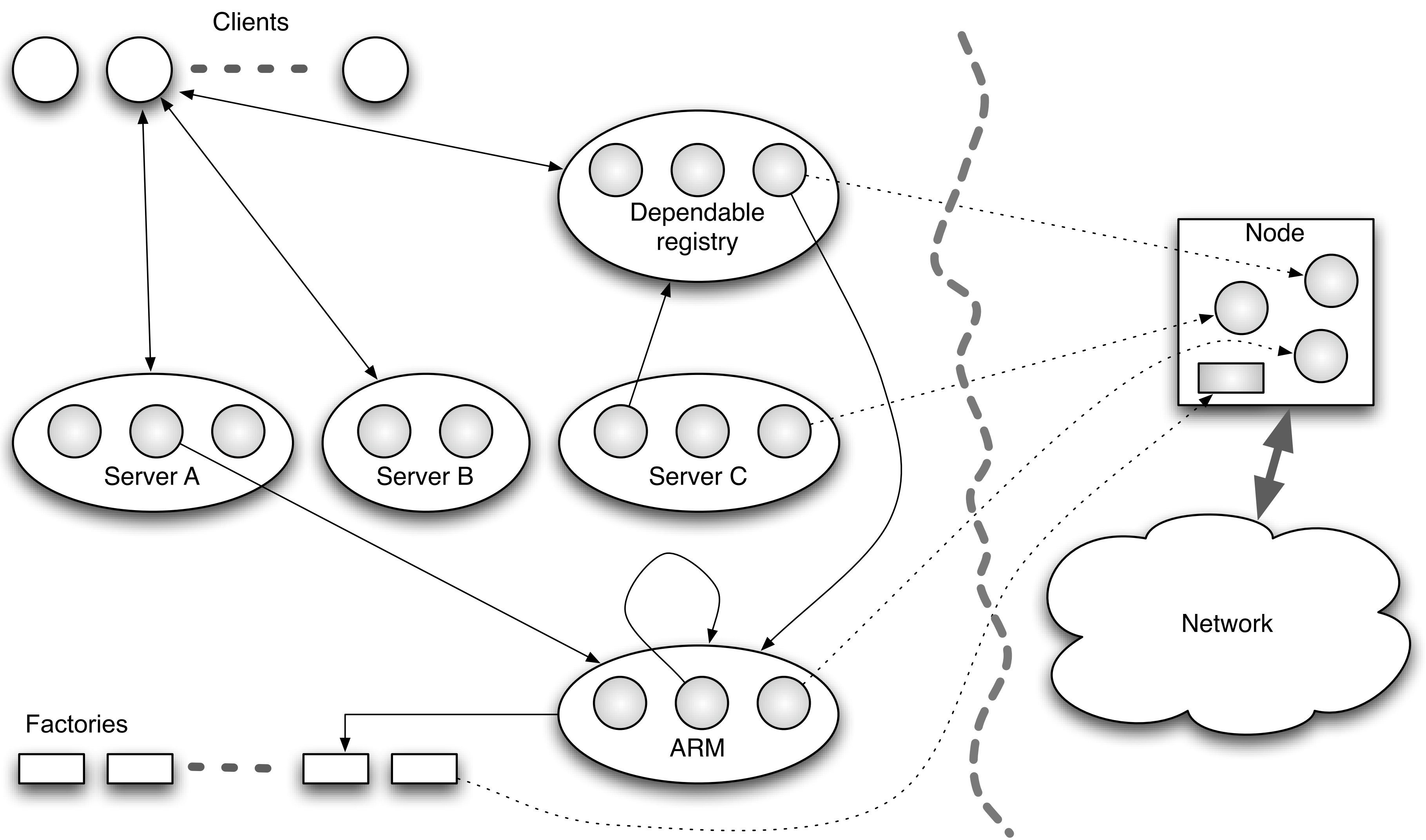
Assumptions

- Pool of processors to host applications
- Replicated stateful applications
- (Wide area network)
- Shared-nothing architecture
  - Neither disk or main memory is shared by processes
  - Avoid distributed file systems
  - State of application must be transmitted across network

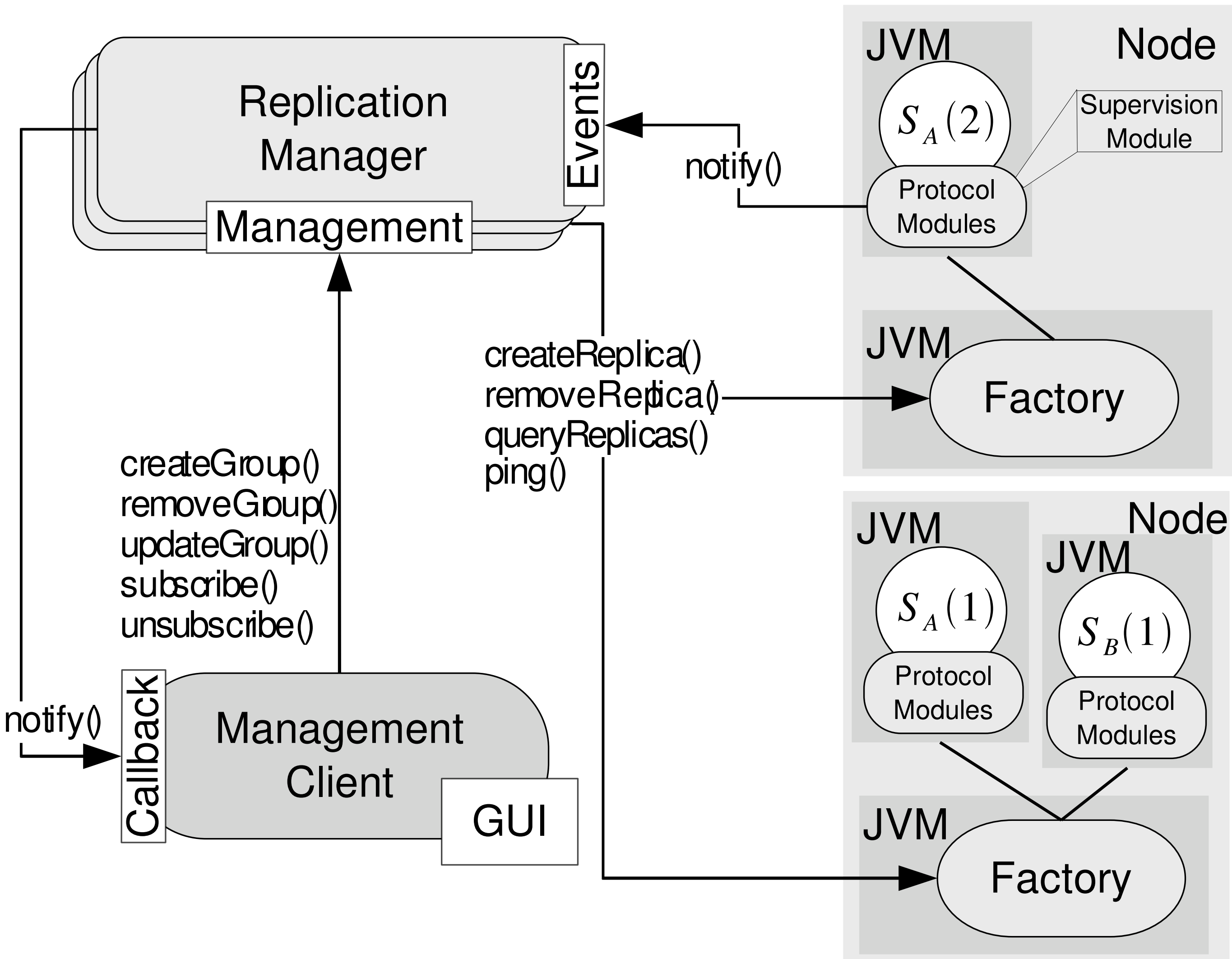
Related work: Centralized Recovery Decisions

- AQuA
  - Leader of group affected by a failure joins the centralized dependability manager to report failure
- FT CORBA
- Jgroup/ARM
  - Report failures to centralized replication manager

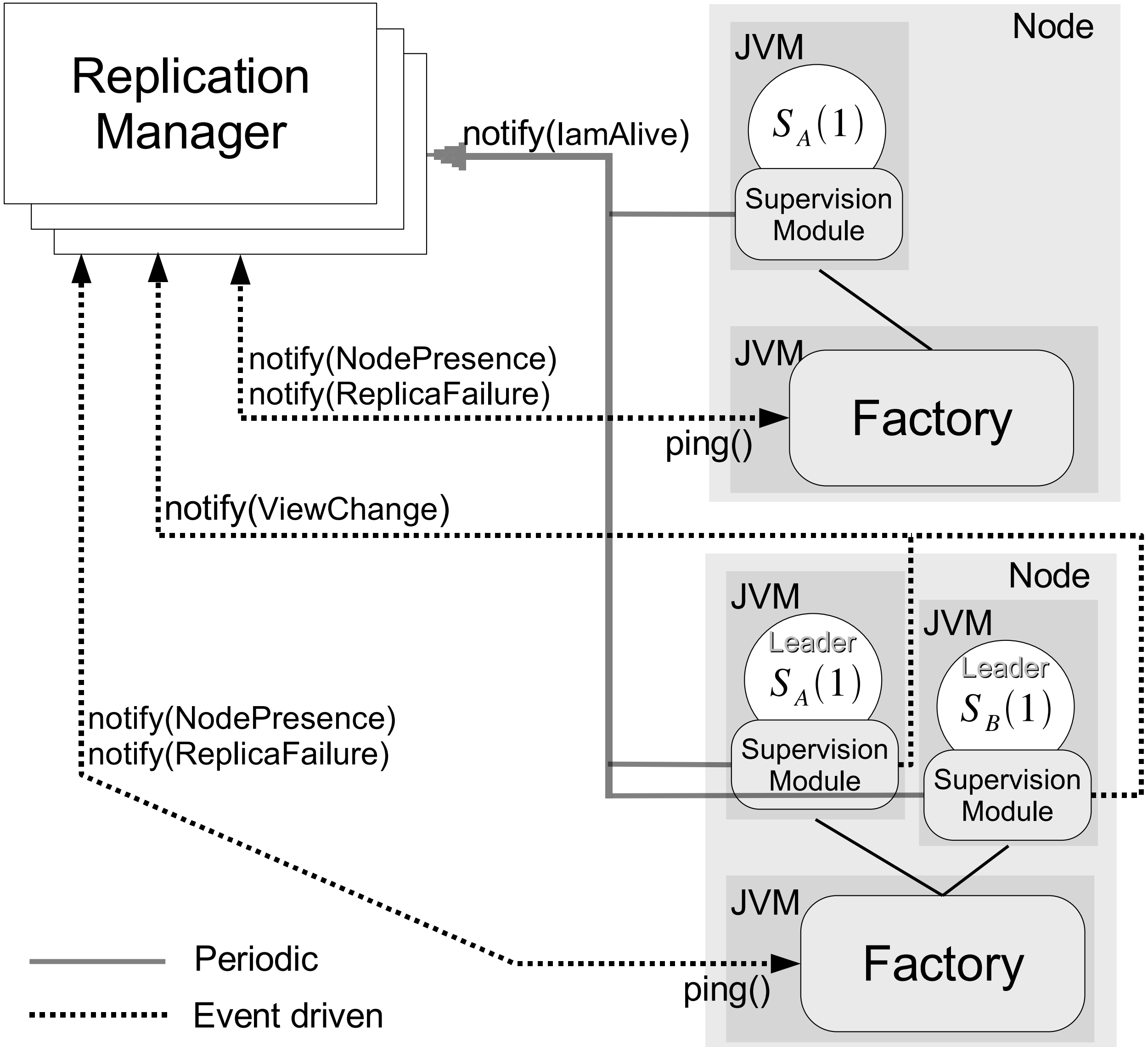
ARM Overview



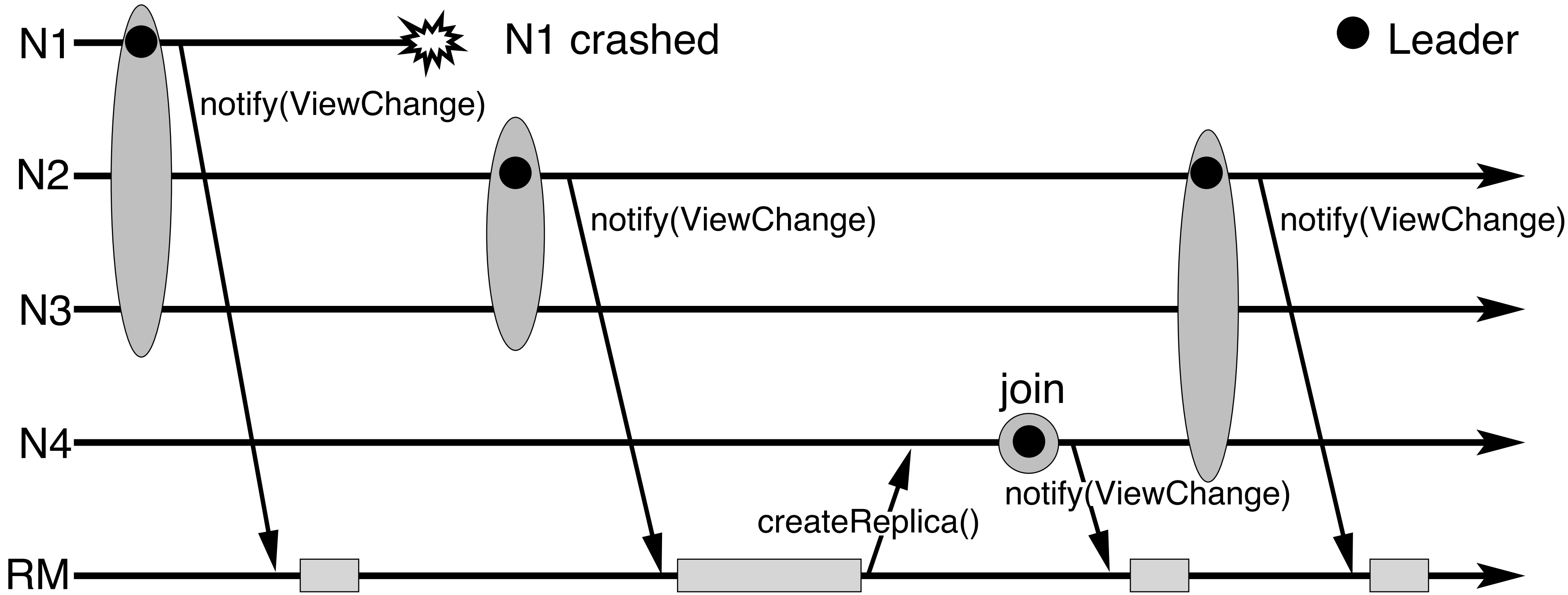
ARM Architecture



Failure Monitoring



Crash Failure and Recovery



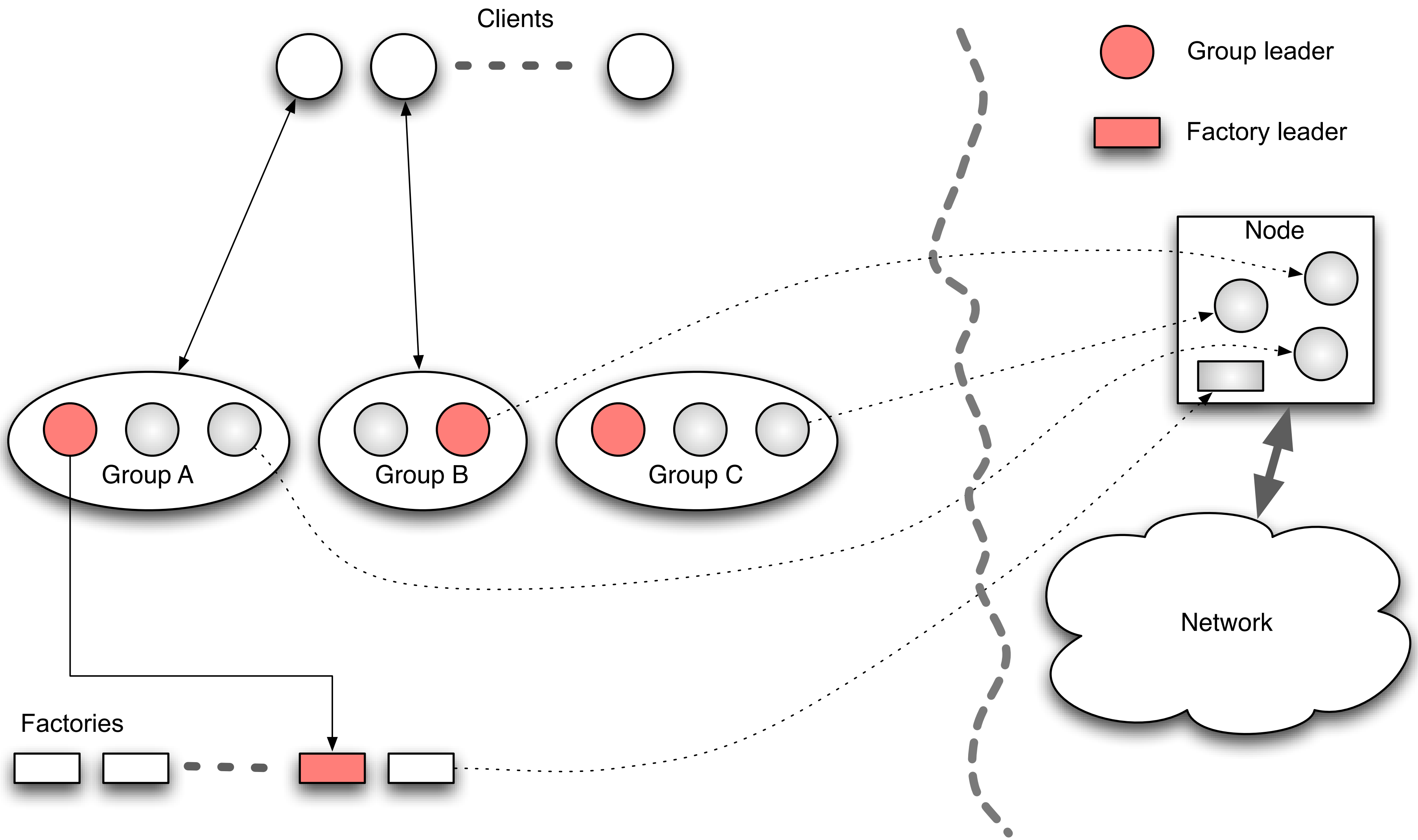
Outline

- Introduction and motivation
- Related work
- Distributed Autonomous Replication Management (DARM)
- Simple Network Partition Evaluation of DARM
- Dependability Evaluation Technique
- Concluding remarks

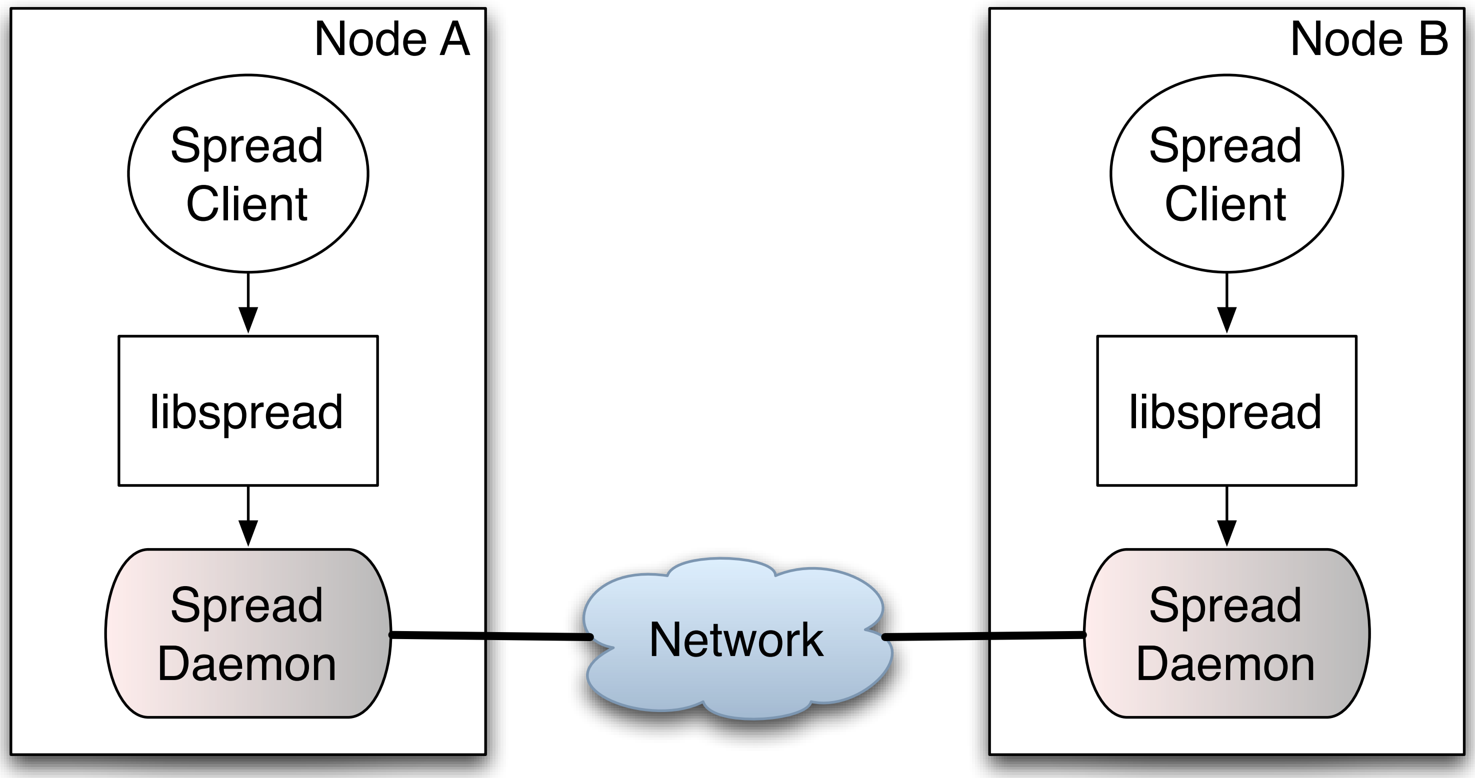
Why go distributed?

- Less infrastructure - less complex
- No need to maintain consistent replicated (centralized) database of deployed groups
- Less communication overhead

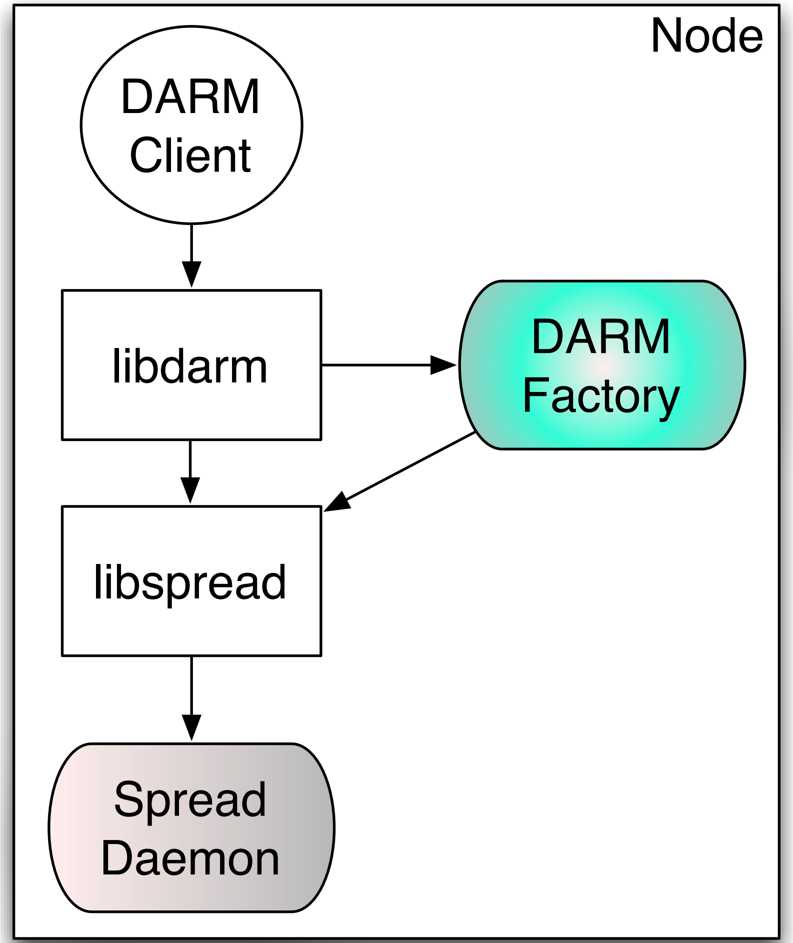
DARM Overview



Spread communication



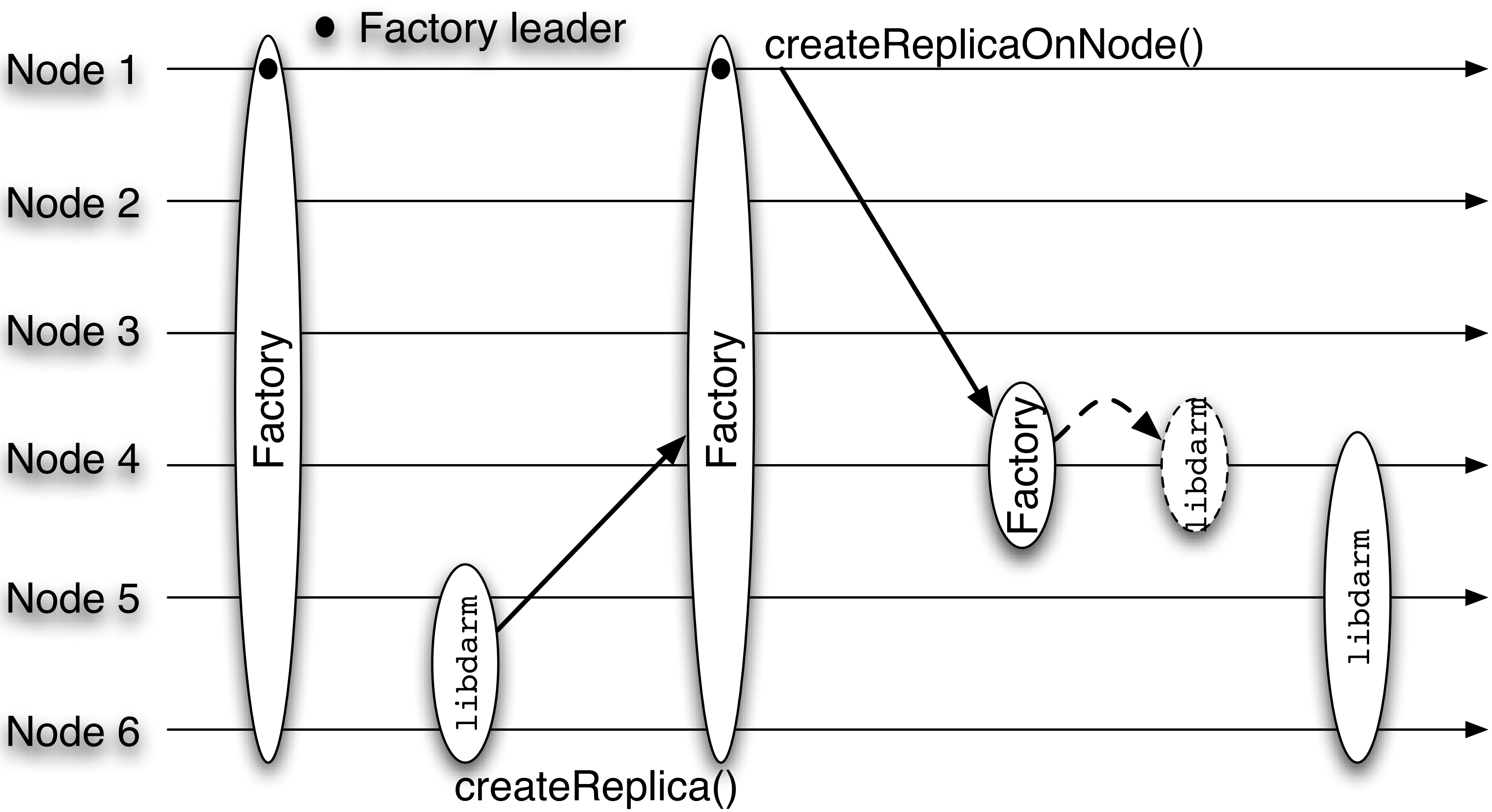
DARM Components



The Factory Group

- Used to install replicas of a given service
- Keeps track of
  - Node availability
  - Local load of nodes
- Interacts with the DARM library
  - To install replacement replicas
- Does not maintain any state about deployed replicas
  - In case of failure: just restart factory to host new replicas

Factory group install replacement replicas



Friday, August 27, 2010

Replica Placement Policy

- *Purpose of replica placement policy:* Describe how replicas should be allocated onto the set of available sites and nodes

1. Find the site with the least # of replicas of the given type
2. Find the node in the candidate site with the least load; ignoring nodes already running the service

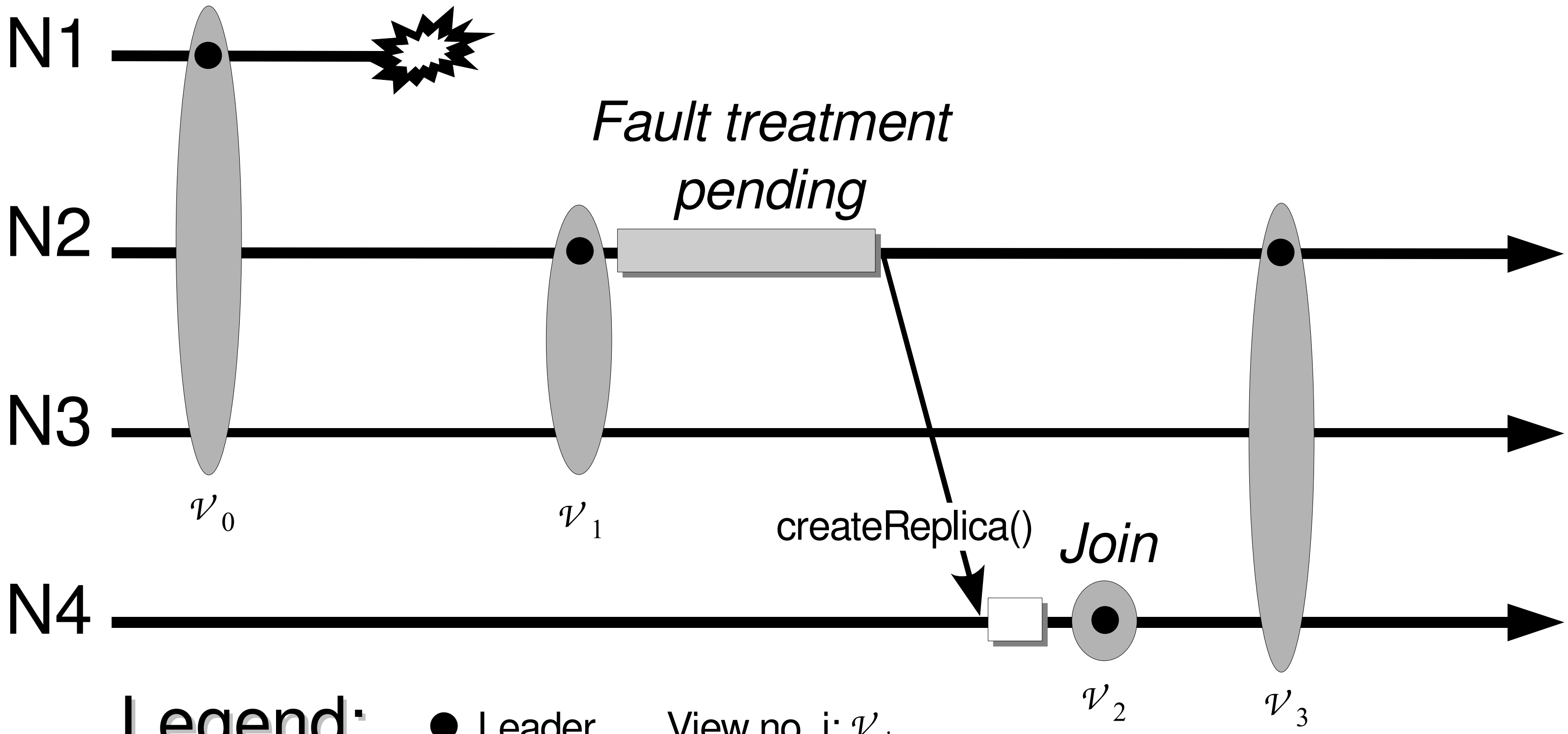
- *Objective of this policy*: Ensure available replicas in each *likely* partition that may arise
  - Avoid collocating two replicas of the same service on the same node
  - Disperse replicas evenly on the available sites
  - Least loaded nodes in each site are selected
  - (Same node may host multiple distinct service types)

Fault Treatment Policy

- KeepMinimalInPartition:
  - Maintain a minimal redundancy level in each partition
- RemovePolicy:
  - Remove excessive replicas
  - Replicas no longer needed to satisfy the fault treatment policy

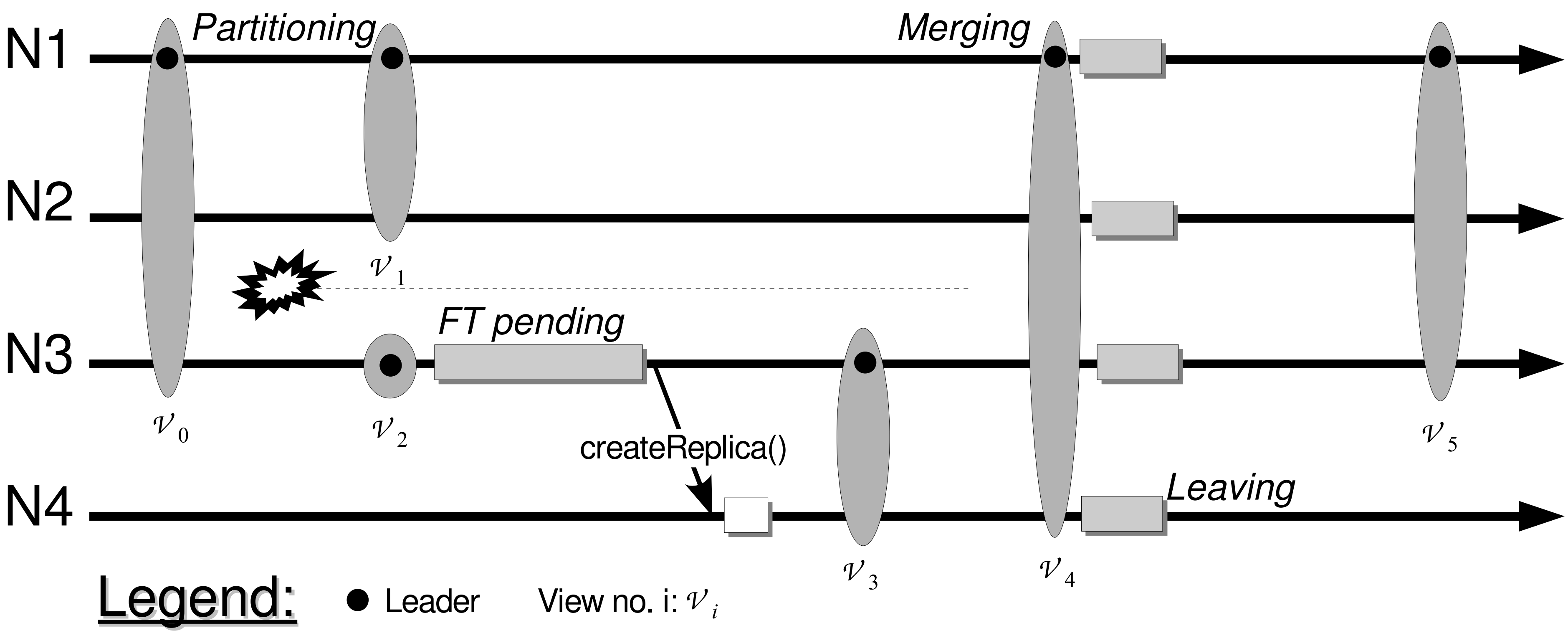
- KeepMinimalInPrimaryPartition:
  - Maintain a minimal redundancy level in the primary partition only
- RedundancyFollowsLoad:
  - Increase redundancy in loaded part of the network

Crash failure-recovery behavior



Legend: ● Leader View no. i: $\mathcal{V}_i$

Failure-recovery with network partitioning and merging



The DARM Library

- libdarm wraps around libspread and intercepts
  - Connection requests to the daemon
    - To verify and finalize runtime configuration of DARM
    - Join DARM private group of the associated application
  - Message receives - SP_receive()
    - If message belongs to DARM private group pass message to DARM
    - Otherwise pass message to application
    - Call SP_receive() again: to avoid having to return control to the application without passing a message
- libdarm also provides functions to set
  - Minimum and maximum number of replicas for the group
  - The recovery and remove delays for the group

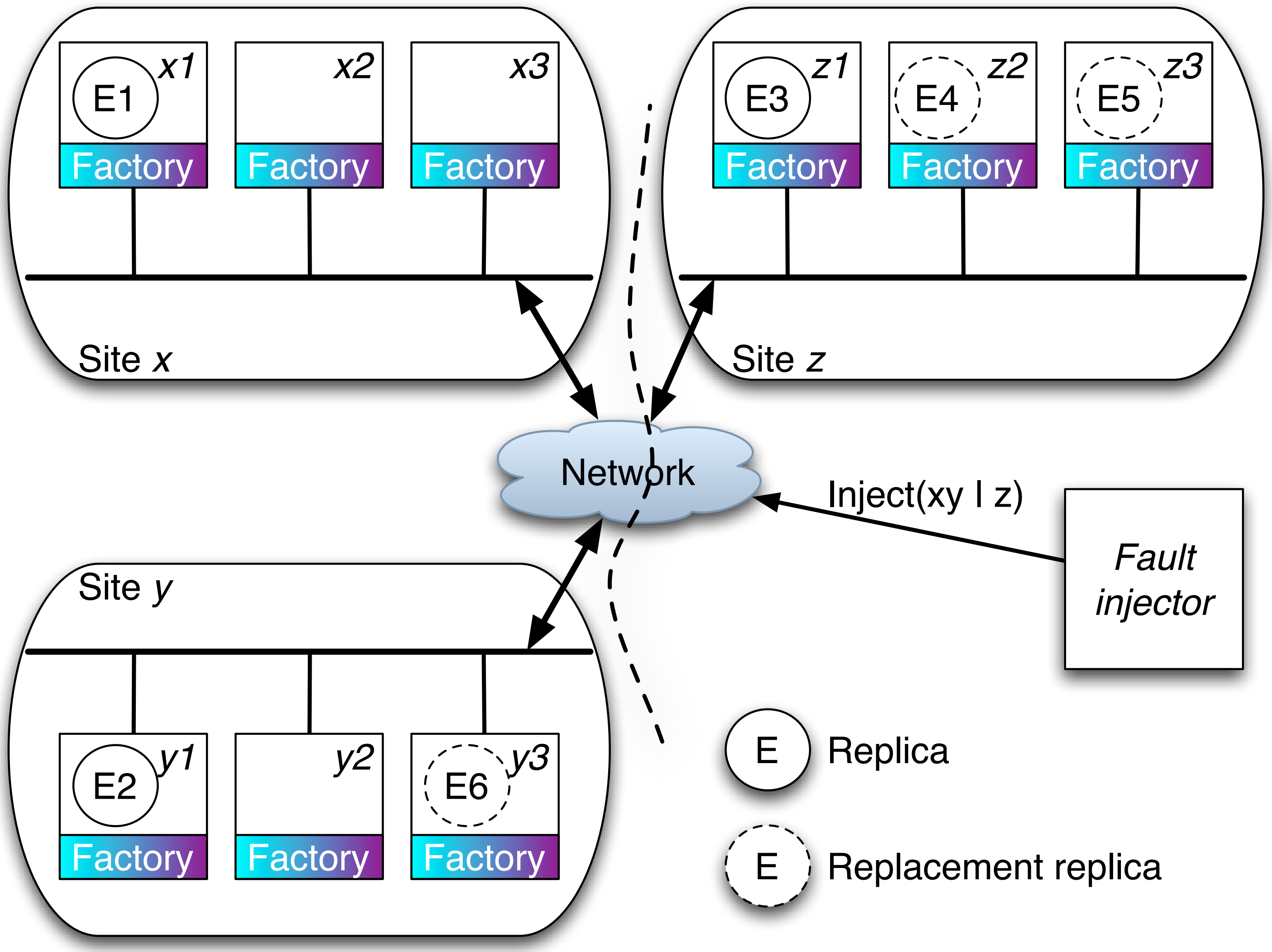
The DARM Library

- Membership messages for the DARM private group
  - Used to decide whether fault treatment is needed
- Bootstrapping applications:
  - Only a single instance of an application needs to be started
  - Assuming the application is configured with some minimum number of replicas
  - DARM will install the required number of replicas

Outline

- Introduction and motivation
- Related work
- Distributed Autonomous Replication Management (DARM)
- Simple Network Partition Evaluation of DARM
- Dependability Evaluation Technique
- Concluding remarks

Target system

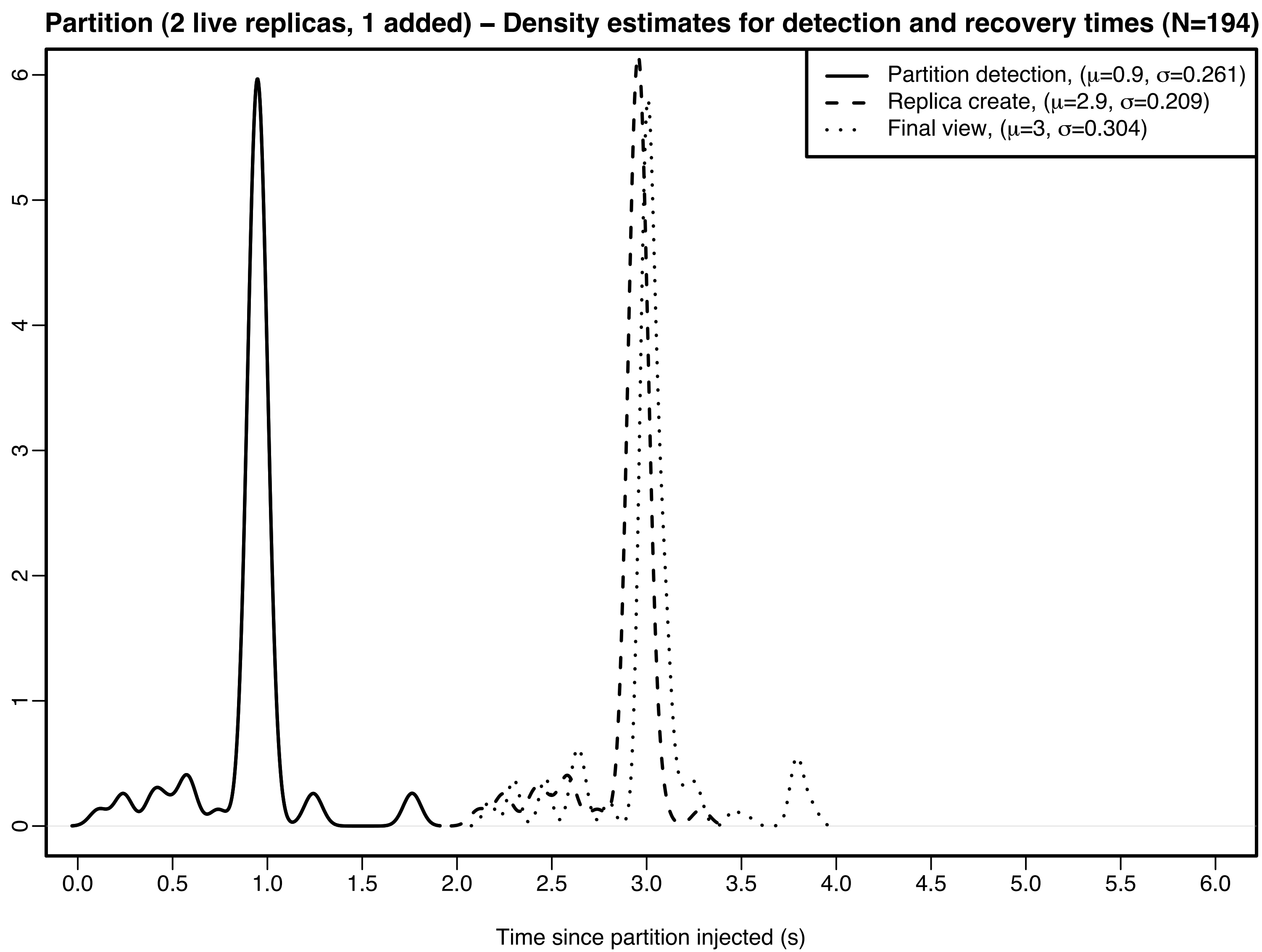


Network Partition/Merge Experiments

- Want to determine
  - the single partition recovery durations
  - corresponding merge of partitions
    (and removal of excessive replicas)

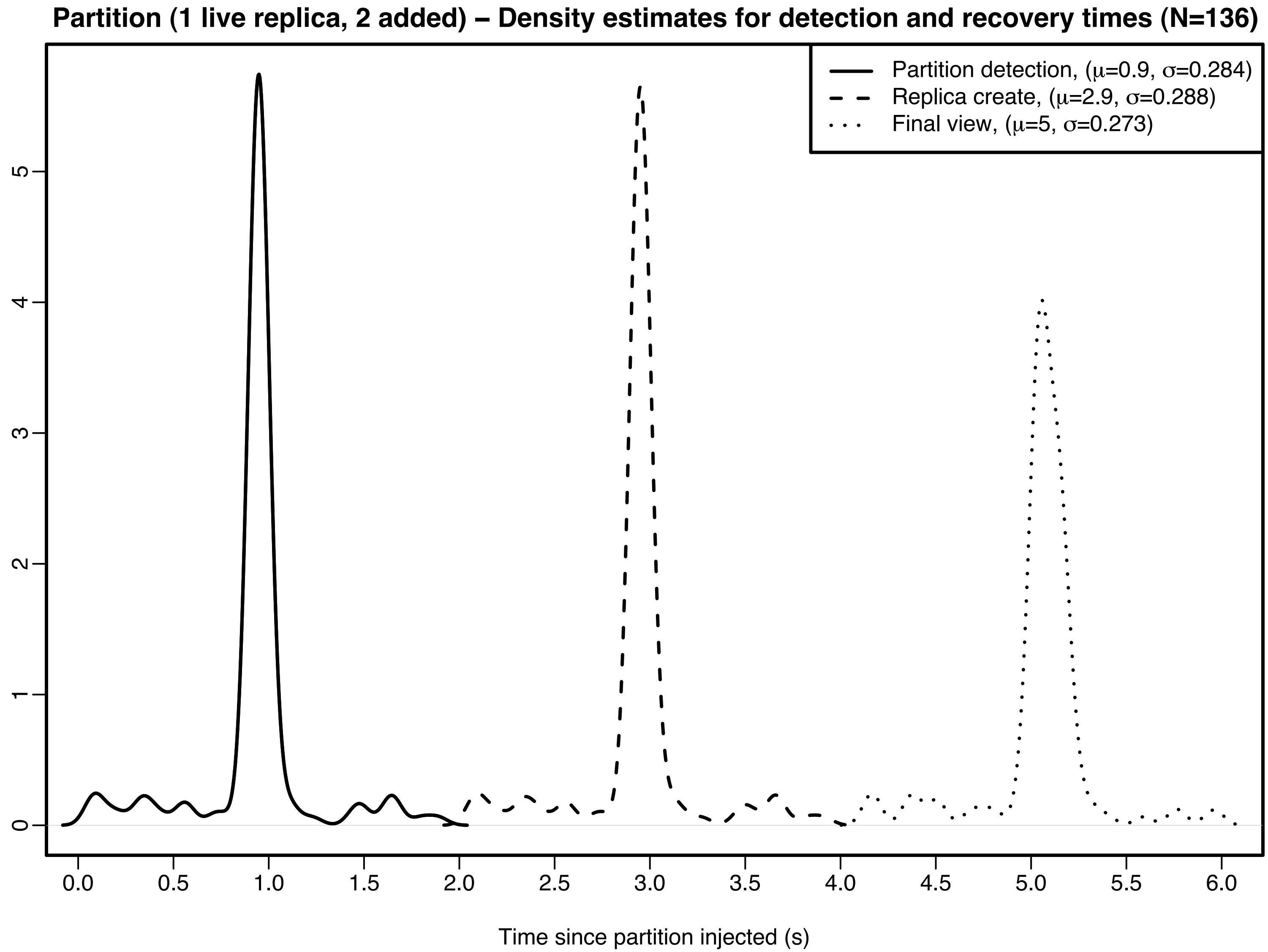
Fast Spread; partition with 2 live replicas

University of Stavanger

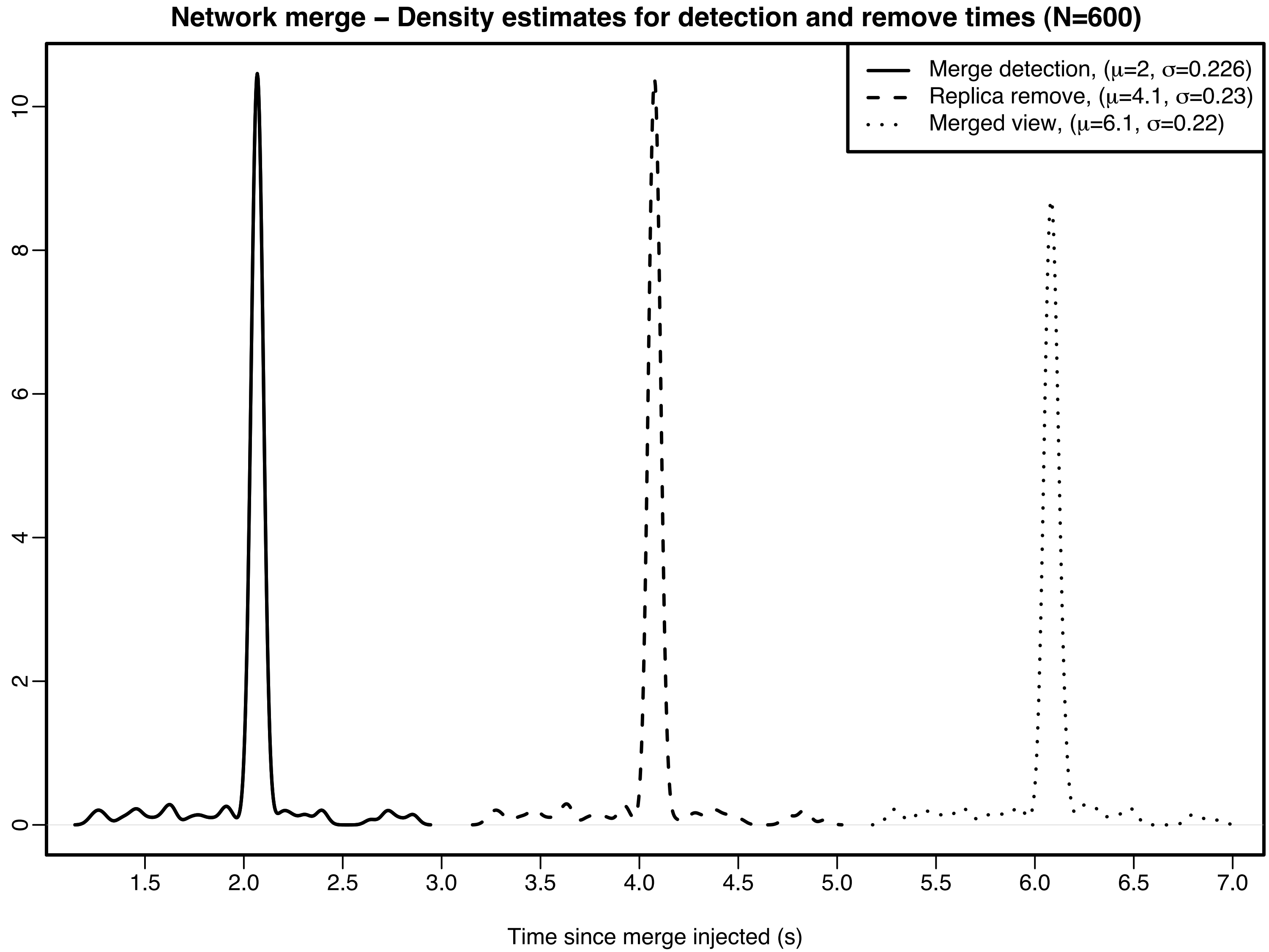


Fast Spread; partition with 1 live replica

University of Stavanger



Fast Spread; Merge, removing 2 replicas



Outline

- Introduction and motivation
- Related work
- Distributed Autonomous Replication Management (DARM)
- Simple Network Partition Evaluation of DARM
- Dependability Evaluation Technique
- Concluding remarks


GLASBERGEN
"It's the latest innovation in office safety. When your computer crashes, an air bag is activated so you won't bang your head in frustration."

Objective of Evaluation

- Provide estimates for dependability attributes:
  - Unavailability
  - System failure intensity
  - Down time

Predicting Dependability Attributes

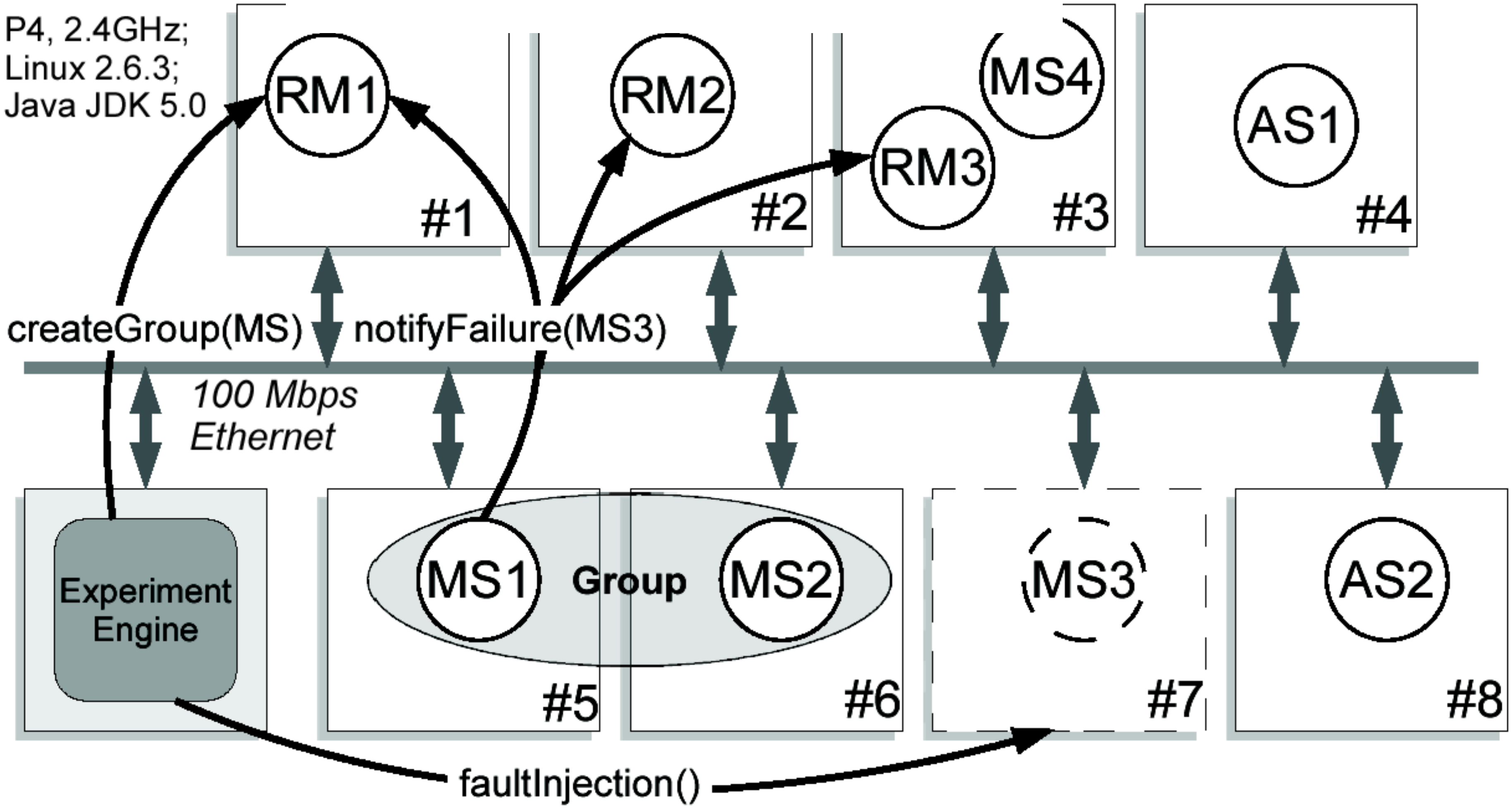
- Use stratified sampling
- Series of lab experiments are performed
  - One or more fault injections in each experiment
    - (all faults manifest themselves as crash failures)
  - According to a homogeneous Poisson process
- Strata := the number of near-coincident failure events
  - A posteriori stratification: Experiments are allocated to different strata after experiment completion
  - Three strata: single, double, and triple failures

Predicting Dependability Attributes

- Offline a posteriori analysis
  - Events are recorded during experiments
  - Used to construct single global timeline of events
  - Compute trajectories on a predefined state machine
- Analysis provide strata classification and various statistics
  - The statistical measures are used as input to estimators for dependability attributes:
    - Unavailability
    - System failure intensity
    - Down time

Target System Illustrated

University of Stavanger

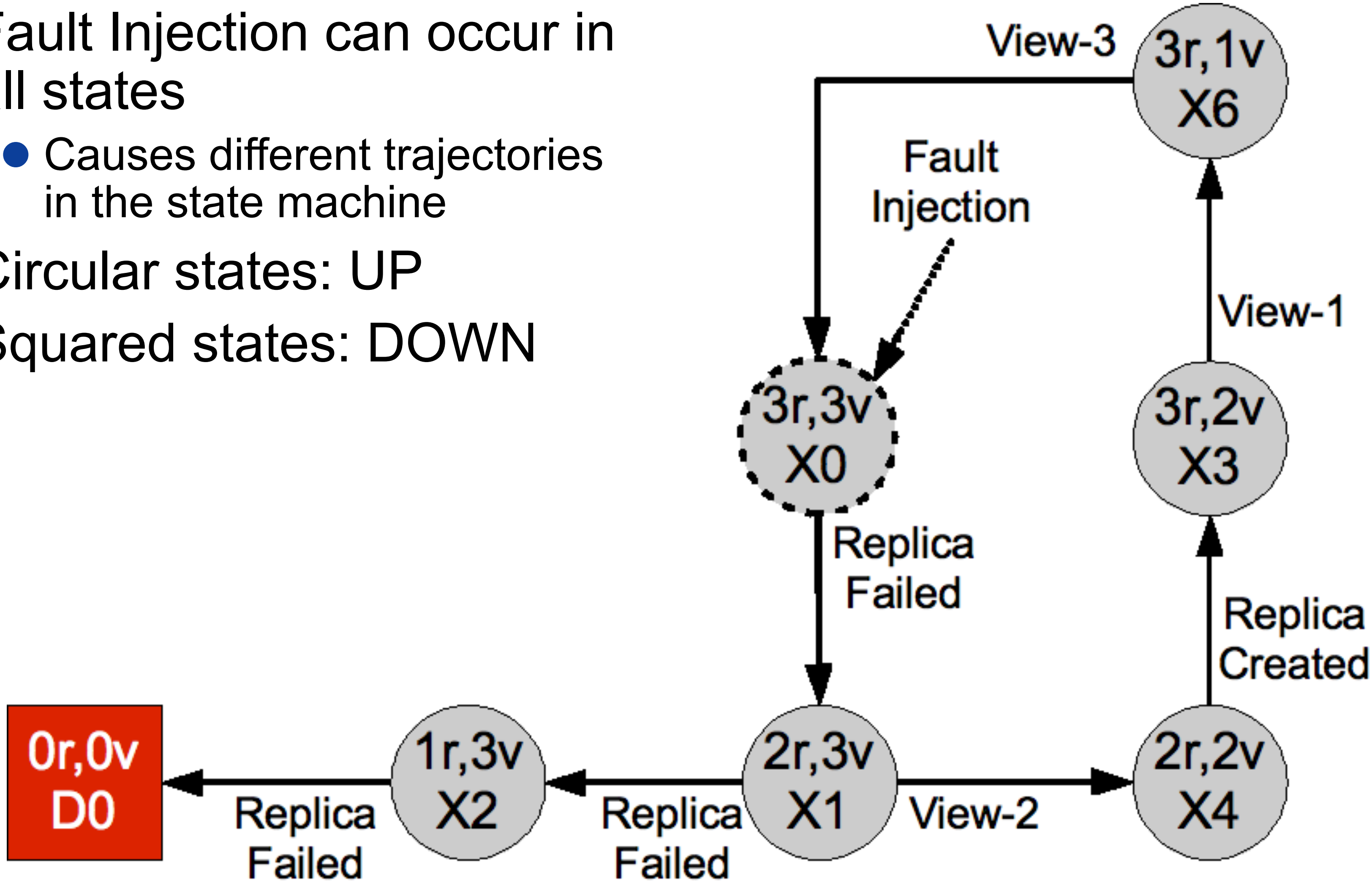


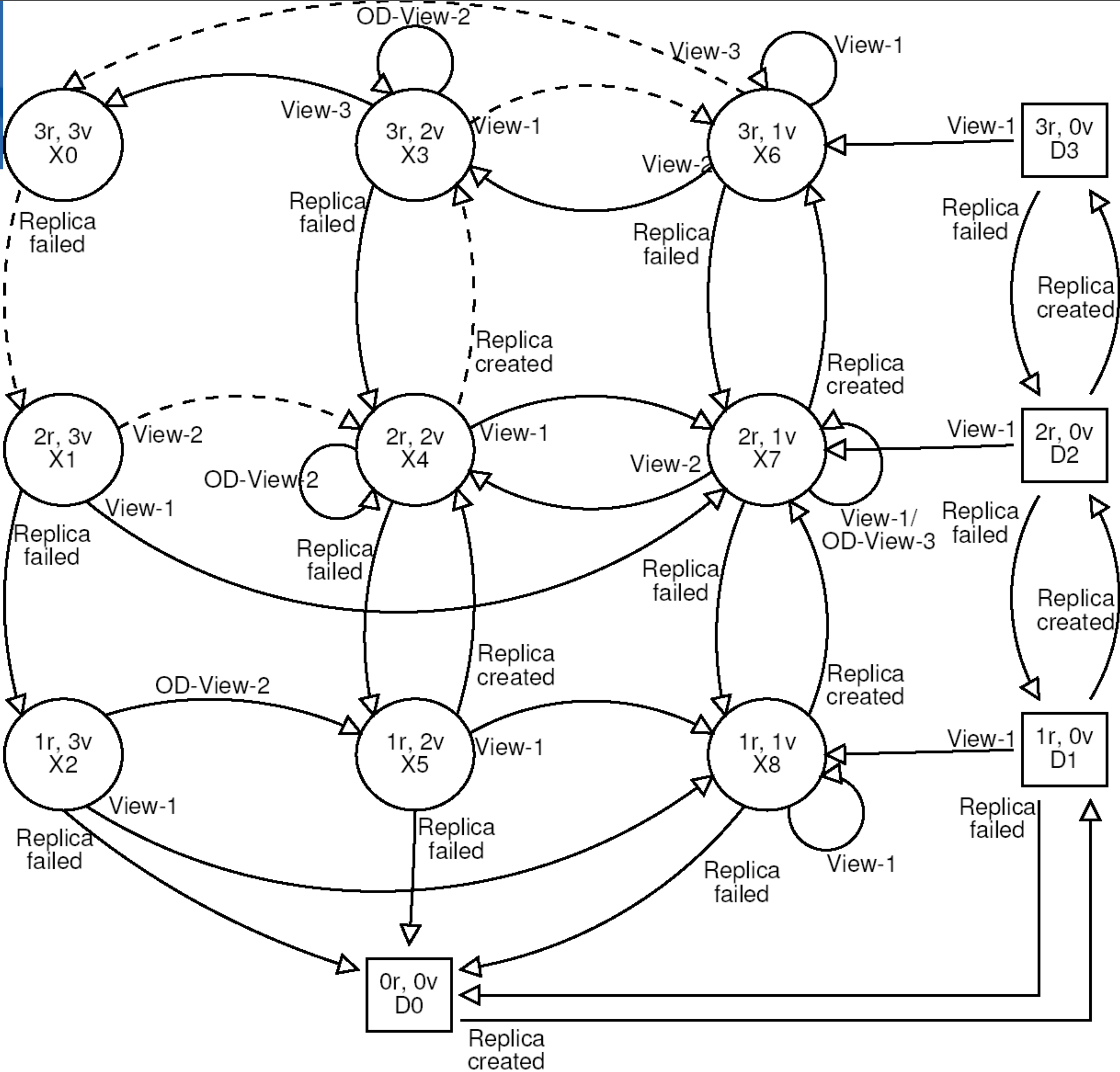
Target System – State Machine

- Failure-recovery behavior of a service
  - Modeled as a state machine (next slide)
  - Events are as seen by the service replicas
- The state machine is only used a posteriori
  - To compute statistics of the experiment
  - (not used to control fault injections)

Partial State Machine

- Fault Injection can occur in all states
  - Causes different trajectories in the state machine
- Circular states: UP
- Squared states: DOWN



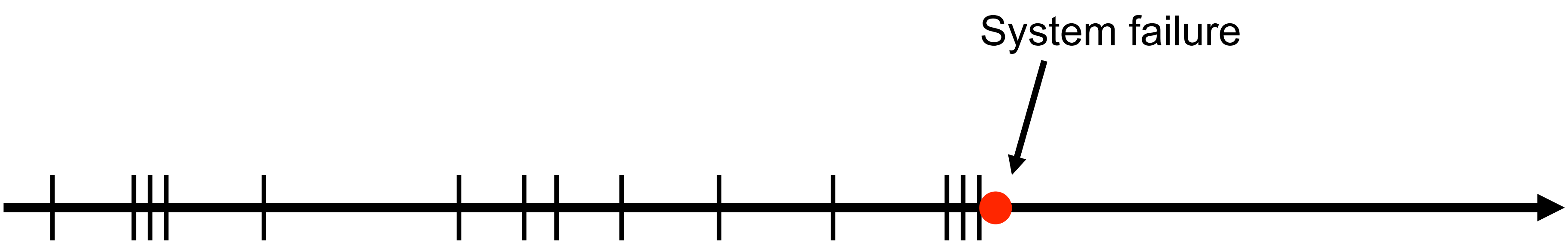

3r, 3v
X0
3r, 2v
X3
3r, 1v
X6
3r, 0v
D3
2r, 3v
X1
2r, 2v
X4
2r, 1v
X7
2r, 0v
D2
1r, 3v
X2
1r, 2v
X5
1r, 1v
X8
1r, 0v
D1
0r, 0v
D0
OD-View-2
View-3
View-1
View-3
View-1
View-2
View-1
Replica failed
Replica created
View-2
OD-View-2
View-1
View-2
View-1/
OD-View-3
OD-View-2
View-1
View-1
View-1
Replica created

Measurement Approach: Timeline of events

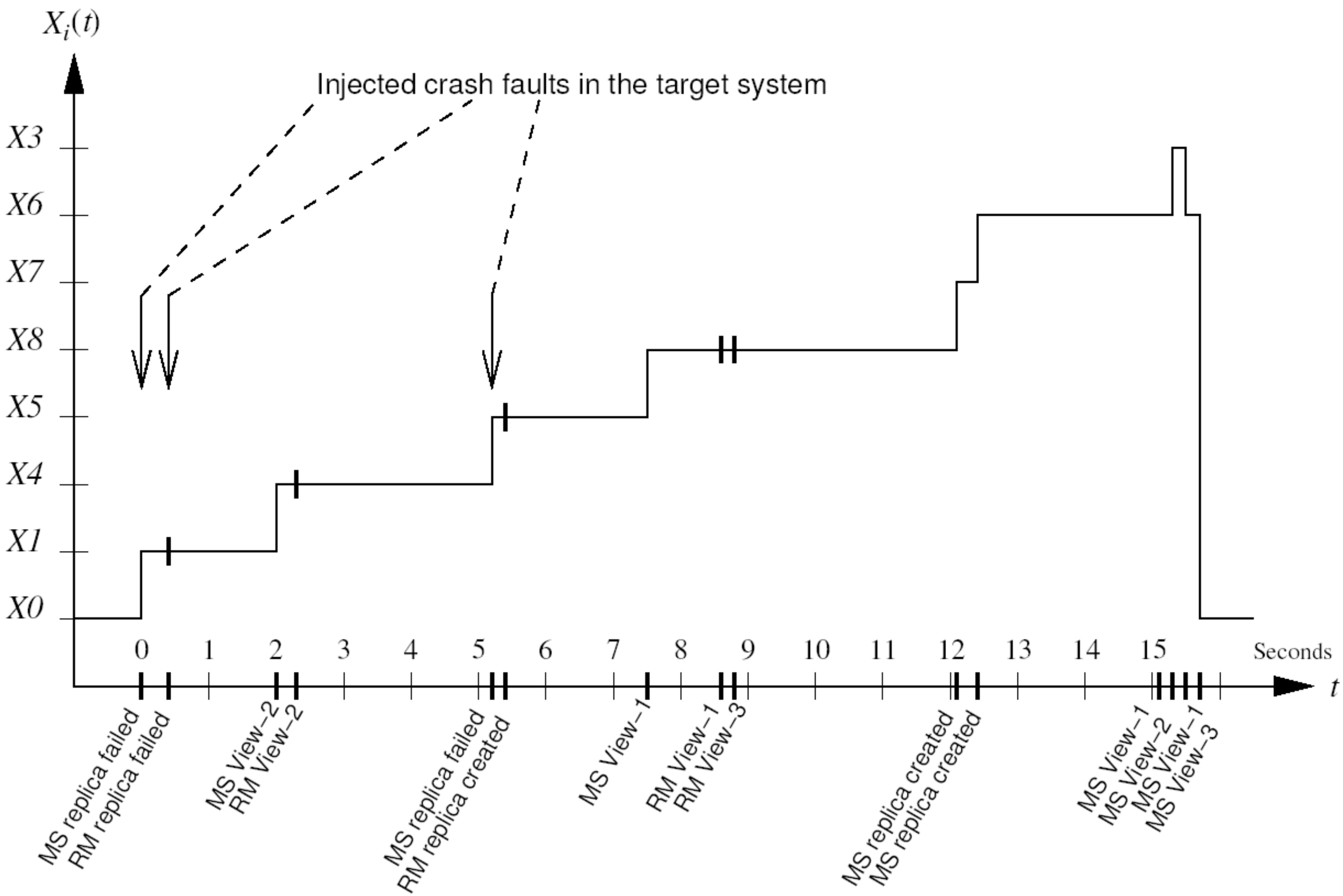
University of Stavanger

- **Place multiple processor failures close together**
  - Examine system behavior of such rare events
  - (determine the rate at which they cause system failure)
  - Use these results to compute *system unavailability*

  - (Given MTBF for a single processor)



The Failure Trajectory



The Failure Trajectory

- Characteristics obtainable from the failure trajectory
  - Unavailability:
    - Down time for trajectory i

$$Y_i^d = g(\underline{X}_i) = \sum_{j=1}^{m_i} I(X_{i_j} \in \mathfrak{F})(t_{i_{j+1}} - t_{i_j})$$

    - Unavailability

$$\hat{U} = \frac{E(Y^d)}{E(Y^d) + (n\lambda)^{-1}} \approx E(Y^d)n\lambda.$$

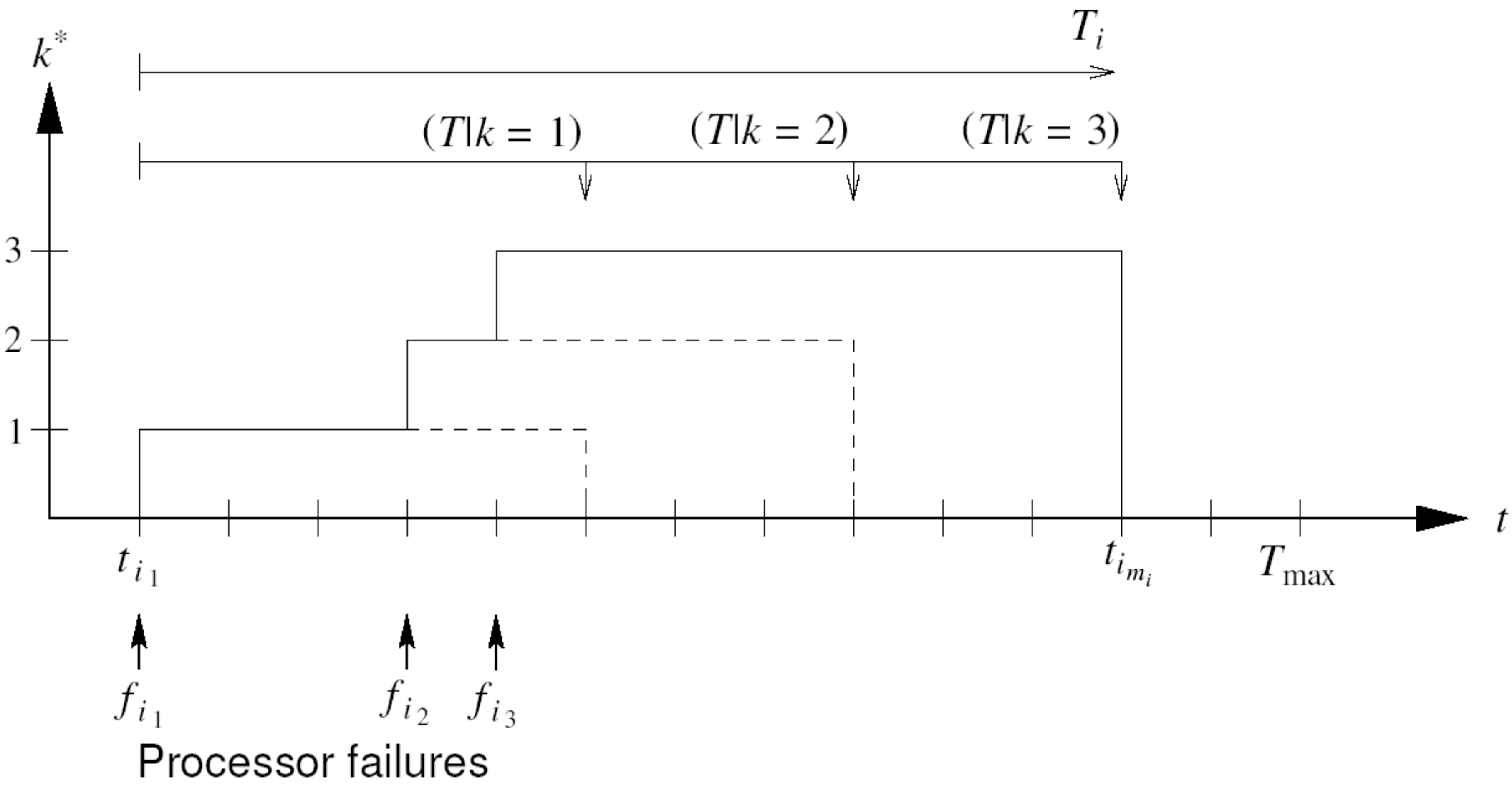
  - Probability of failure (reliability)
    - (formulas in the paper)

Experimental Strategy

- Consider multiple near-coincident failures
- Classify experiments into strata S_k
  - If k failure events occurred in the trajectory
- Each strata sampled separately
- Collected samples for each stratum
  - Can obtain statistics for the system in that stratum
  - E.g., the expected duration of a stratum S_k trajectory:

$$\Theta_k = E(T|S_k) \text{ and } \sigma_k = Var(T|S_k)$$

Sampling Scheme



Estimators

- In real systems, failure intensity λ very low;
  - i.e, $\lambda^{-1} >> Tmax$
  - π_k = probability of a trajectory reaching stratum S_k

$$\pi_k = \sum_{\forall i \in S_k} p_i$$

- Unconditional probability of a sample in
  - Stratum S_2

$$\pi_2 = (n-1)\lambda\Theta_1\pi_1$$

  - Stratum S_3
    - (in the paper)

Experimental Results

- Perform fault injections on target system according to sampling scheme
- 3000 (lab) experiments performed
  - Aiming for 1000 in each stratum
  - Classified as stratum S_k if exactly k failures occur before completion of experiment

ARM (top) / DARM (bottom)

University of Stavanger

Table 1. Results obtained from the experiments (in milliseconds).

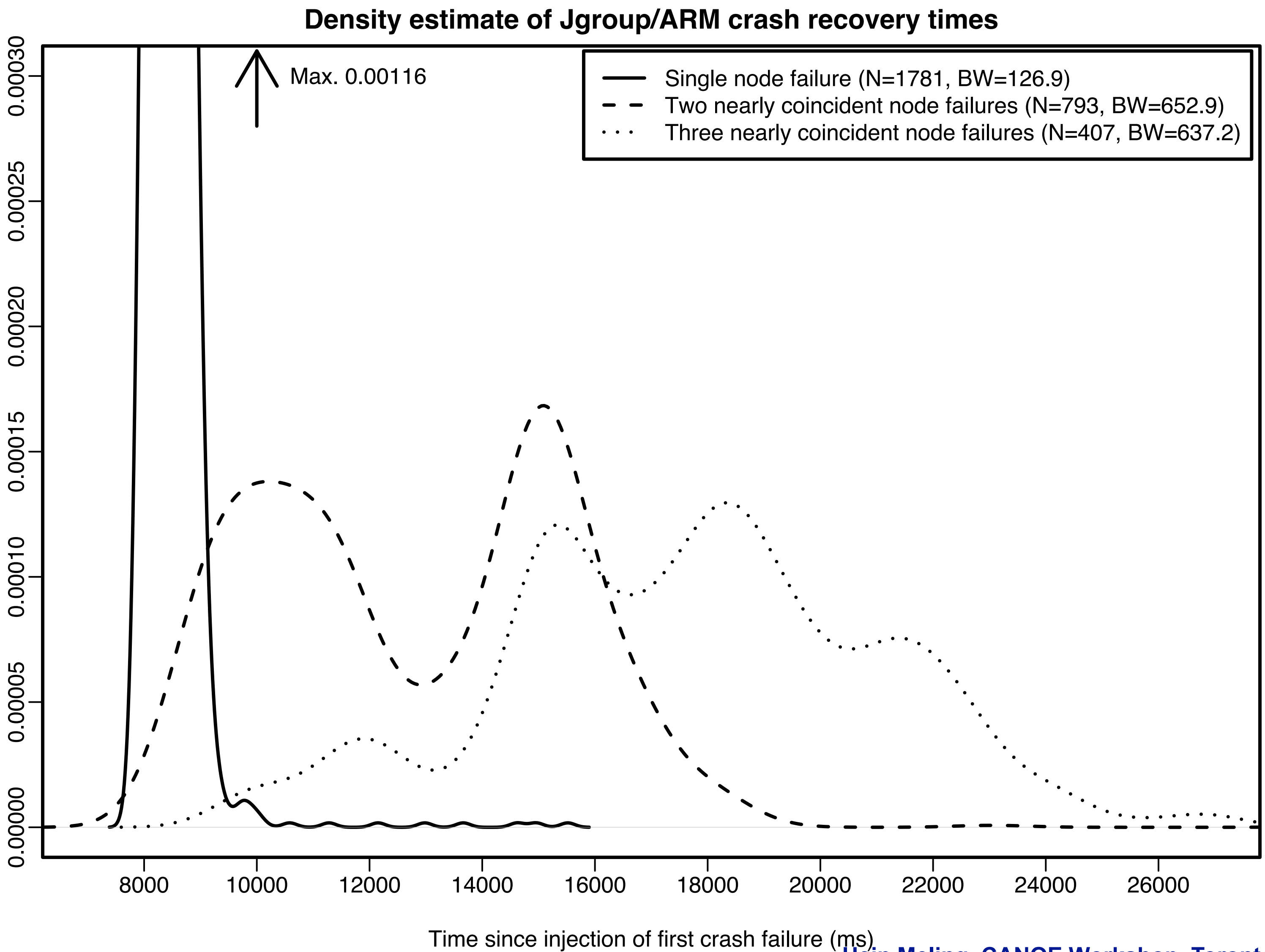
Classification	Count	$\Theta_k = E(T\|S_k)$	sd=$\sqrt{\sigma_k}$	Θ_k, 95% conf.int.
Stratum S_1	1781	8461.77	185.64	(8328.98, 8594.56)
Stratum S_2	793	12783.91	1002.22	(12067.01, 13500.80)
Stratum S_3	407	17396.55	924.90	(16734.96, 18058.13)

Classification	Count	$\theta_k = E(T\|S_k)$	sd $= \sqrt{\sigma_k}$	θ_k, 95% conf.int.	Highest	Lowest
$Strata_1$	2265	2569.22	478.23	(1631.89, 3506.55)	16659	1742
$Strata_2$	591	4158.83	1039.10	(2122.18, 6195.47)	12869	2496
$Strata_3$	110	5966.58	1550.90	(2926.82, 9006.35)	16086	3046

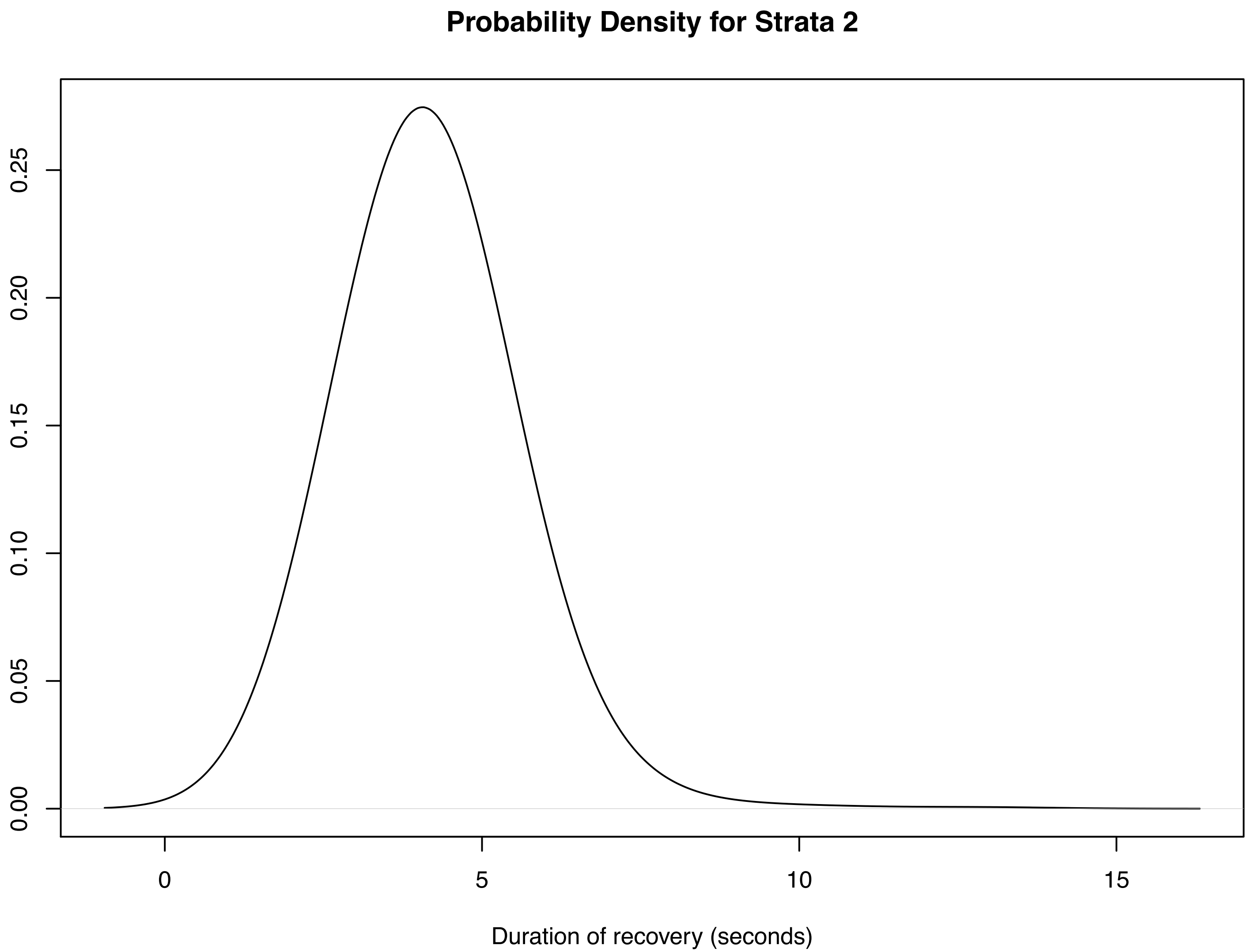
Experimental Results

- 19 experiments (0.63%) were classified as inadequate
  - 16 experiments failed to recover
  - 3 experiments experienced additional not-intended failures
  - Of the 16, two were for S1, 6 for S2 and 11 for S3
  - These 16 are due to deficiencies in Jgroup/ARM
- These inadequate runs are accounted for as trajectories visiting a down state for 5 minutes (typically a reboot)
- For DARM there were 2 inadequate experiments

Prob. Density Function



Prob. Density S_2 (DARM)



Applying the Equations

University of Stavanger

Table 2. Computed probabilities, unavailability metric and the system MTBF.

	Experiment Recovery Period		Processor Recovery (5 min.)		Manual Processor Recovery (2 hrs.)	
	Processor Mean Time Between Failure (MTBF=λ^{-1}) (in days)					
	100	200	100	200	100	200
π_1	0.99999314	0.99999657	0.99975688	0.99987845	0.99412200	0.99707216
π_2	$6.855602 \cdot 10^{-6}$	$3.427801 \cdot 10^{-6}$	$2.430555 \cdot 10^{-4}$	$1.215278 \cdot 10^{-4}$	$5.833333 \cdot 10^{-3}$	$2.916667 \cdot 10^{-3}$
π_3	$4.072921 \cdot 10^{-11}$	$1.018230 \cdot 10^{-11}$	$5.595341 \cdot 10^{-8}$	$1.398835 \cdot 10^{-8}$	$4.466146 \cdot 10^{-5}$	$1.116536 \cdot 10^{-5}$
$\hat{U}$	$4.671318 \cdot 10^{-7}$	$2.335617 \cdot 10^{-7}$	$2.777102 \cdot 10^{-4}$	$1.388720 \cdot 10^{-4}$	$6.627480 \cdot 10^{-3}$	$3.323574 \cdot 10^{-3}$
$\hat{\Lambda}^{-1}$	20.3367 yrs	40.6741 yrs	-	-	-	-

	Experiment Recovery Period		Processor Recovery (5 min.)		Manual Processor Recovery (2 hrs.)	
	Processor Mean Time Between Failure (pmtbf=λ^{-1}) (in days)					
	100	200	100	200	100	200
π_1	0.9999979184	0.9999989592	0.9997568889	0.9998784583	0.9941238281	0.9970726237
π_2	$2.0815438 \cdot 10^{-6}$	$1.0407719 \cdot 10^{-6}$	$2.4305555 \cdot 10^{-4}$	$1.2152777 \cdot 10^{-4}$	$5.8333333 \cdot 10^{-3}$	$2.9166666 \cdot 10^{-3}$
π_3	$4.0903937 \cdot 10^{-12}$	$1.0225984 \cdot 10^{-12}$	$5.5447048 \cdot 10^{-8}$	$1.3861762 \cdot 10^{-8}$	$4.2838541 \cdot 10^{-5}$	$1.0709635 \cdot 10^{-5}$
$\hat{U}$	$4.1317108 \cdot 10^{-17}$	$5.1646385 \cdot 10^{-18}$	$2.7771024 \cdot 10^{-4}$	$1.3887200 \cdot 10^{-4}$	$6.6274921 \cdot 10^{-3}$	$6.6471508 \cdot 10^{-3}$
$\hat{\Lambda}^{-1}$	212 yrs	851 yrs	-	-	-	-

Concluding Remarks

University of Stavanger

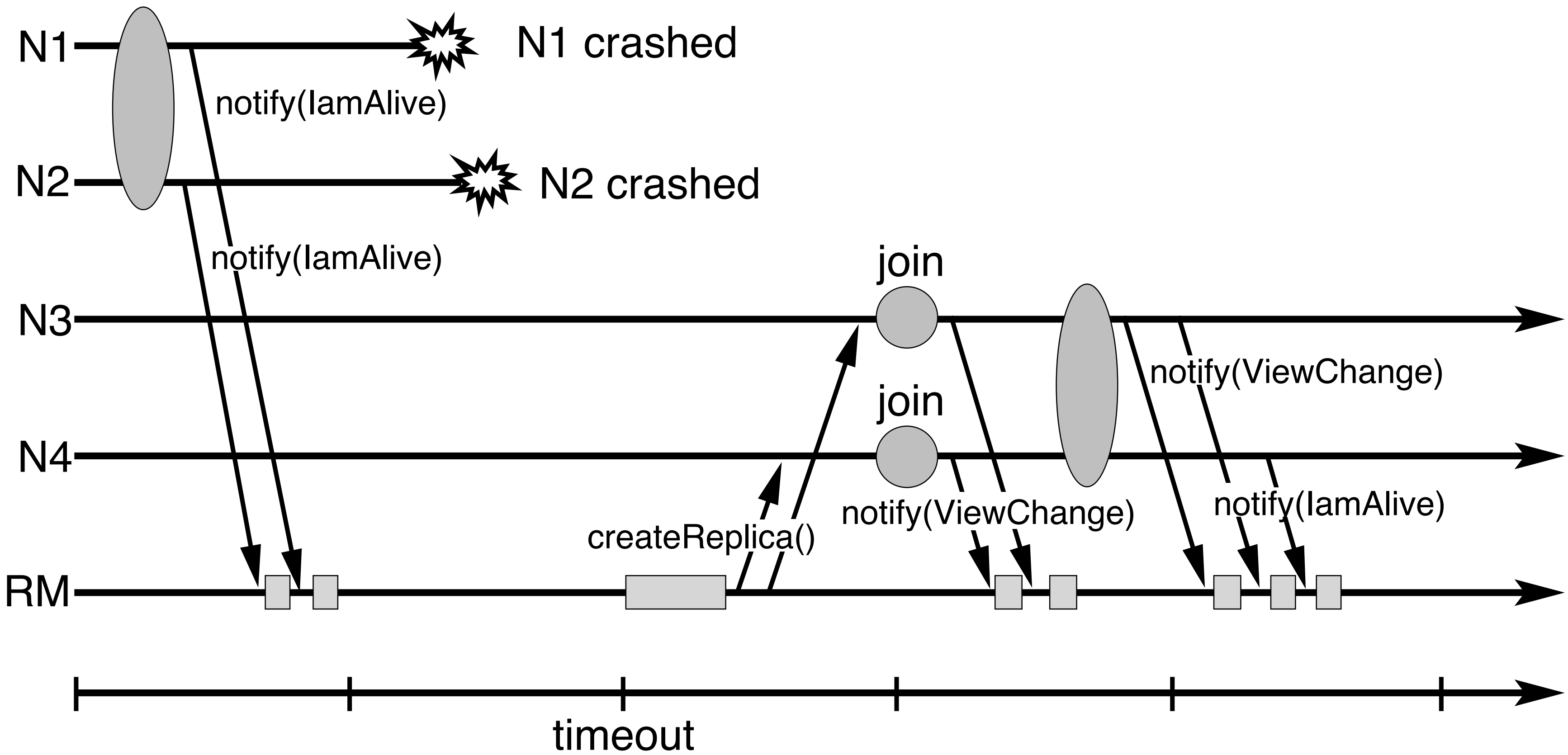
- DARM supports autonomous fault treatment
  - Recovery decisions are *distributed* to the individual groups
  - In previous systems recovery decisions were centralized
    - Complex and error-prone
- DARM has been released as open source at:
  - darm.ux.uis.no
- We are performing more advanced measurements
  - Client perceived availability
  - Longer executions and with other parameters to get statistically significant results
- Experimental results indicate that self-repairing systems can obtain very high availability and MTBF
- Automated fault injection tool
  - Proved very useful for uncovering a number of subtle bugs
  - Allows for systematic stress and regression testing

Open Issues

University of Stavanger

- Handling full group failures
  - ARM have a centralized component to monitor all groups
  - DARM only monitors the group from within itself
  - Could let the factory handle this in some way
    - Lease/Renew or simple pinging
- Management tasks to simplify deployment of applications
  - Self-configuration
  - Reconfiguration of nodes that can host replicas
- Express policies in terms of equations
- Implement more policies

Group Failure Handling



University of Stavanger

Thanks!

References

[1] Hein Meling, Alberto Montresor, Bjarne E. Helvik, and Ozalp Babaoglu. Jgroup/ARM: a distributed object group platform with autonomous replication management. *Software: Practice and Experience*, 38(9):885-923, July 2008.

[2] Hein Meling and Joakim L. Gilje. A Distributed Approach to Autonomous Fault Treatment in Spread. In *Proceedings of the 7th European Dependable Computing Conference* (EDCC). IEEE Computer Society, May 2008.

[3] Bjarne E. Helvik, Hein Meling, and Alberto Montresor. An Approach to Experimentally Obtain Service Dependability Characteristics of the Jgroup/ARM System. In *Proceedings of the Fifth European Dependable Computing Conference* (EDCC), volume 3463 of Lecture Notes in Computer Science, pages 179-198. Springer-Verlag, April 2005.

[4] Hein Meling. Adaptive Middleware Support and Autonomous Fault Treatment: Architectural Design, Prototyping and Experimental Evaluation. PhD thesis, Norwegian University of Science and Technology, Department of Telematics, May 2006.