An Architecture for Versatile Dependability

Tudor Dumitraş and Priya Narasimhan

Electrical and Computer Engineering Department
Carnegie Mellon University
USA

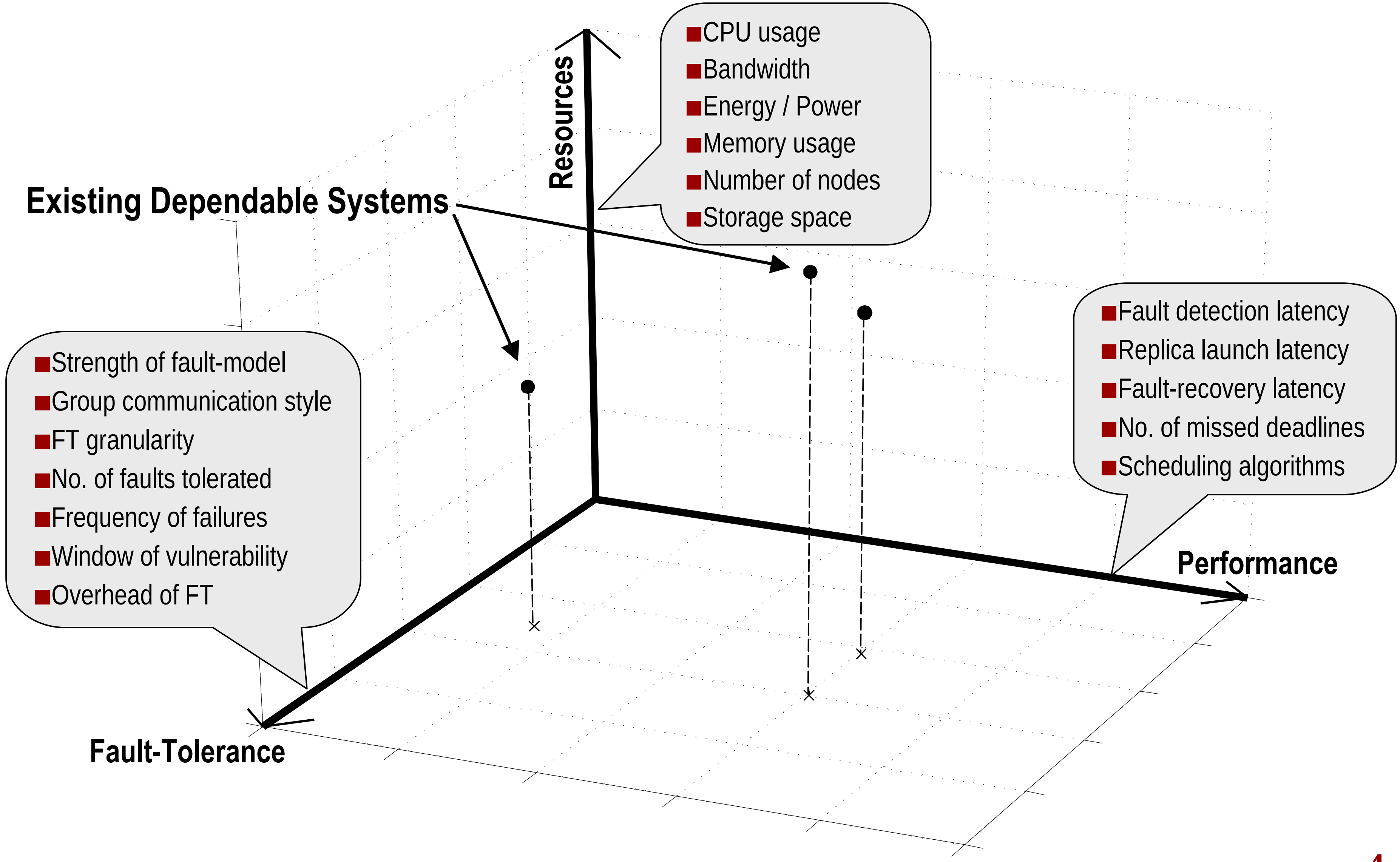
Motivation

- **The requirements of dependable systems are often *conflicting***
  - Example: meeting deadlines in the presence of faults
    - Meeting deadlines requires a predictable system, while faults are inherently unpredictable!
- **These conflicts must be seen as a *trade-off***
  - Usually, dependable systems hard-code such trade-offs in their design choices
- **Architectures should become tunable to provide support for:**
  - Configuring the system before deployment
  - Adapting to changes in the environment during run-time
  - Maintaining the system throughout its life-cycle

Outline

- Versatile Dependability
- An Architecture for Versatile Dependability
- Case Study: Tuning the System Scalability
- Conclusions
- Future Work

Versatile Dependability

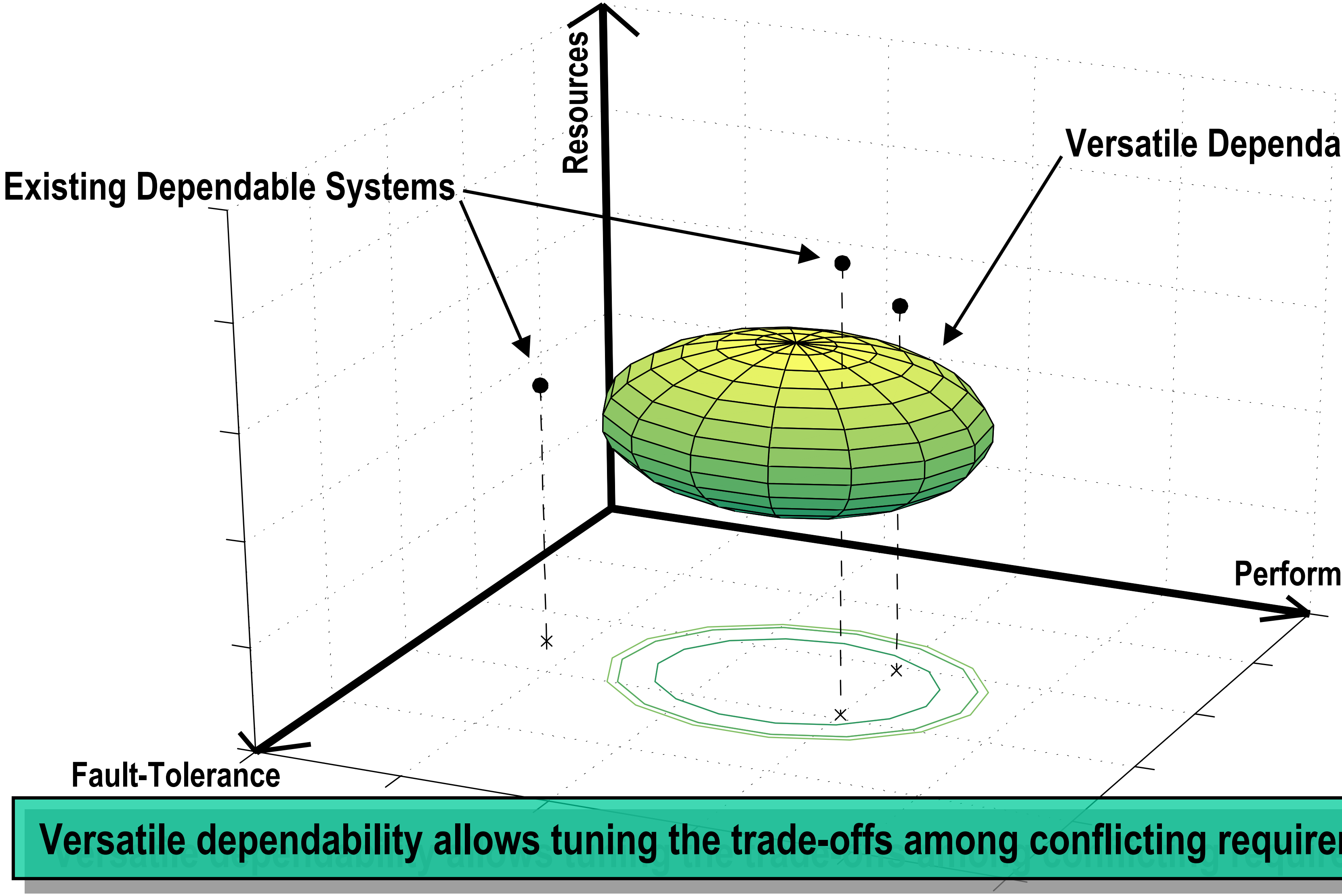


- CPU usage
- Bandwidth
- Energy / Power
- Memory usage
- Number of nodes
- Storage space

- Strength of fault-model
- Group communication style
- FT granularity
- No. of faults tolerated
- Frequency of failures
- Window of vulnerability
- Overhead of FT

- Fault detection latency
- Replica launch latency
- Fault-recovery latency
- No. of missed deadlines
- Scheduling algorithms

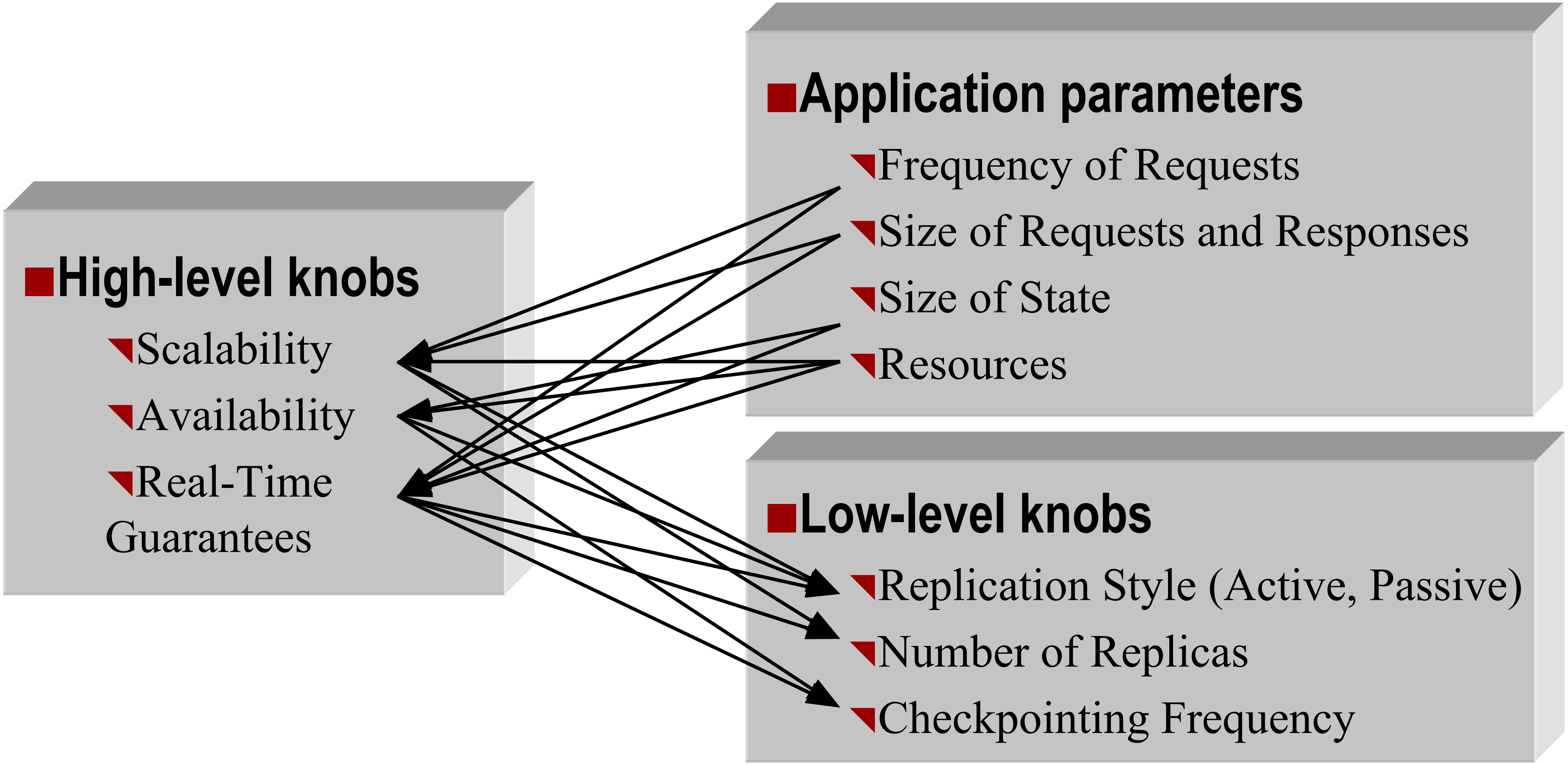
Versatile Dependability



Versatile dependability allows tuning the trade-offs among conflicting requirements.

Versatile Dependability: Control Knobs

- Versatile dependability provides control *knobs* to tune the trade-offs

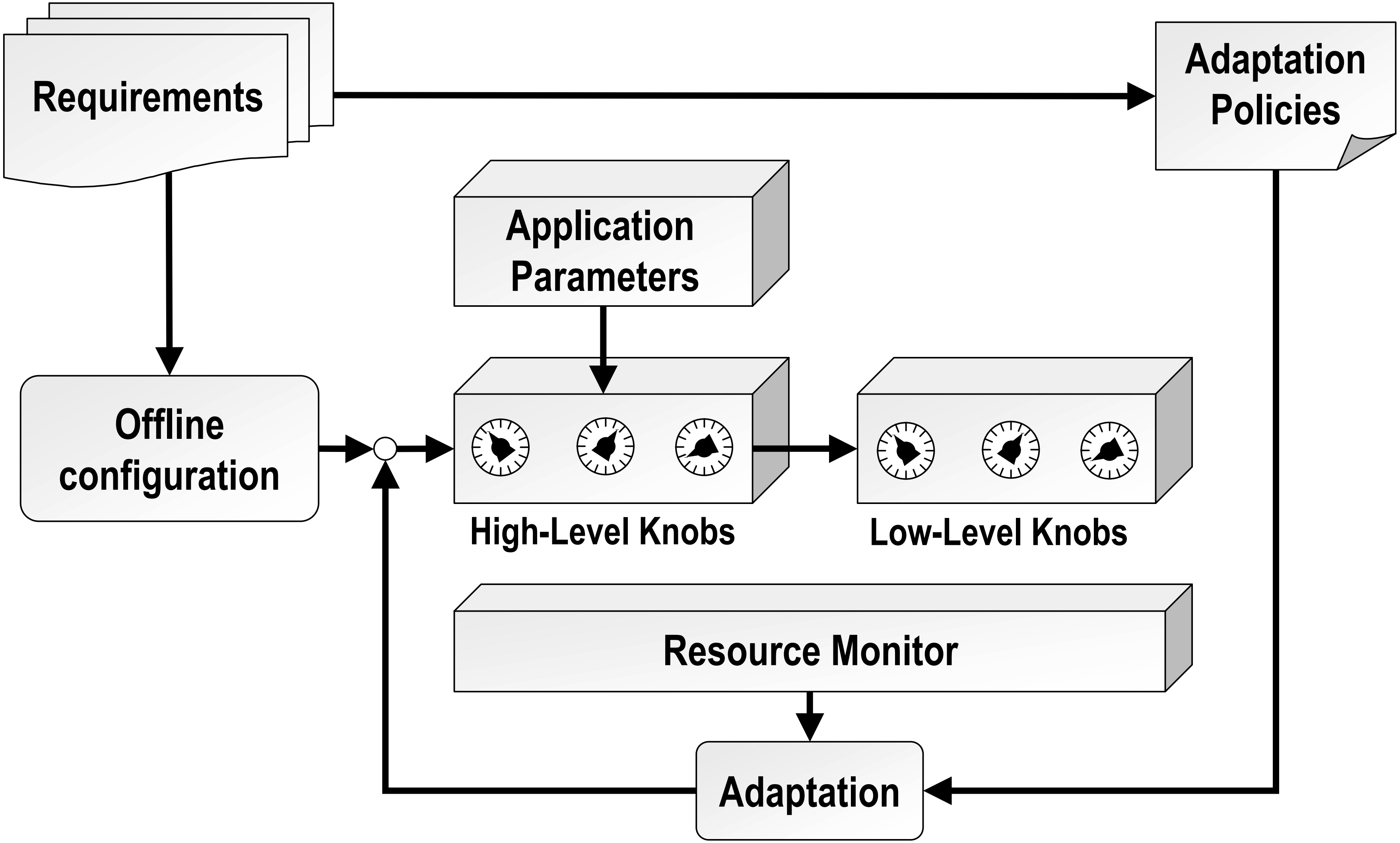


- **High-level knobs**
  - Scalability
  - Availability
  - Real-Time Guarantees

- **Application parameters**
  - Frequency of Requests
  - Size of Requests and Responses
  - Size of State
  - Resources

- **Low-level knobs**
  - Replication Style (Active, Passive)
  - Number of Replicas
  - Checkpointing Frequency

Versatile Dependability Loop



Architecture for Versatile Dependability

- Design goals:
  - *Tunability and homogeneity*: one infrastructure, multiple knobs
  - *Quantifiability*: using precise metrics to evaluate trade-offs
  - *Transparency*: support for fault-tolerance unaware applications
  - *Ease of use*: simple knobs that are intuitively easy to adjust

Architecture for Versatile Dependability

- **Distributed software architecture**
  - No central point
  - Tunable redundancy levels
  - Components work independently and synchronize using group communication
- **Enhancement to CORBA middleware**
- **Part of the MEAD Project**

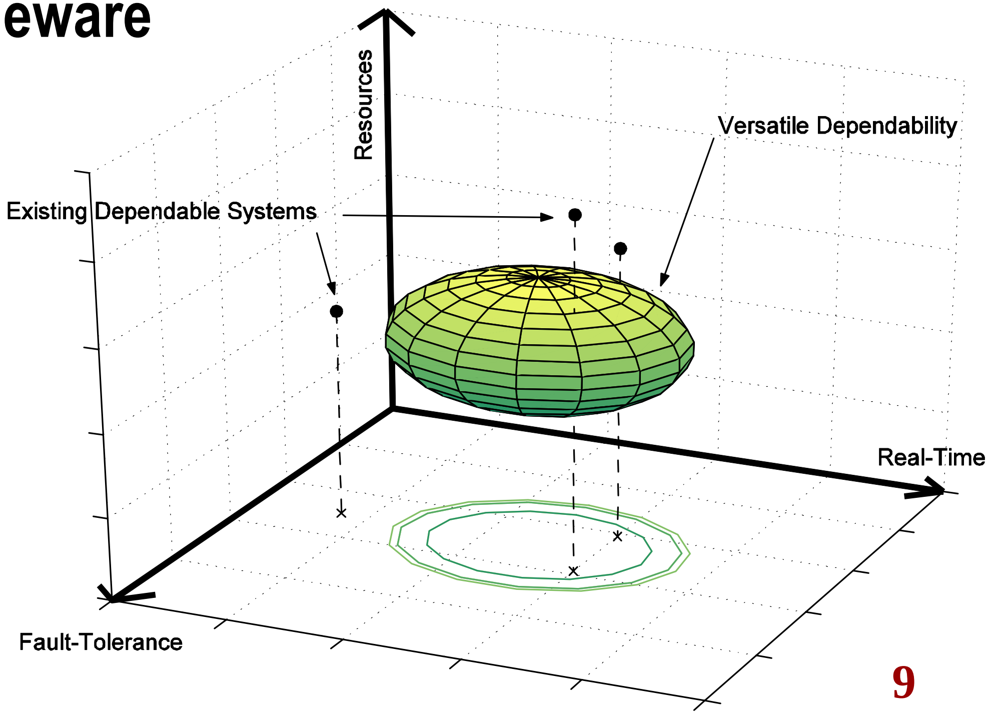
  Middleware for

  Embedded

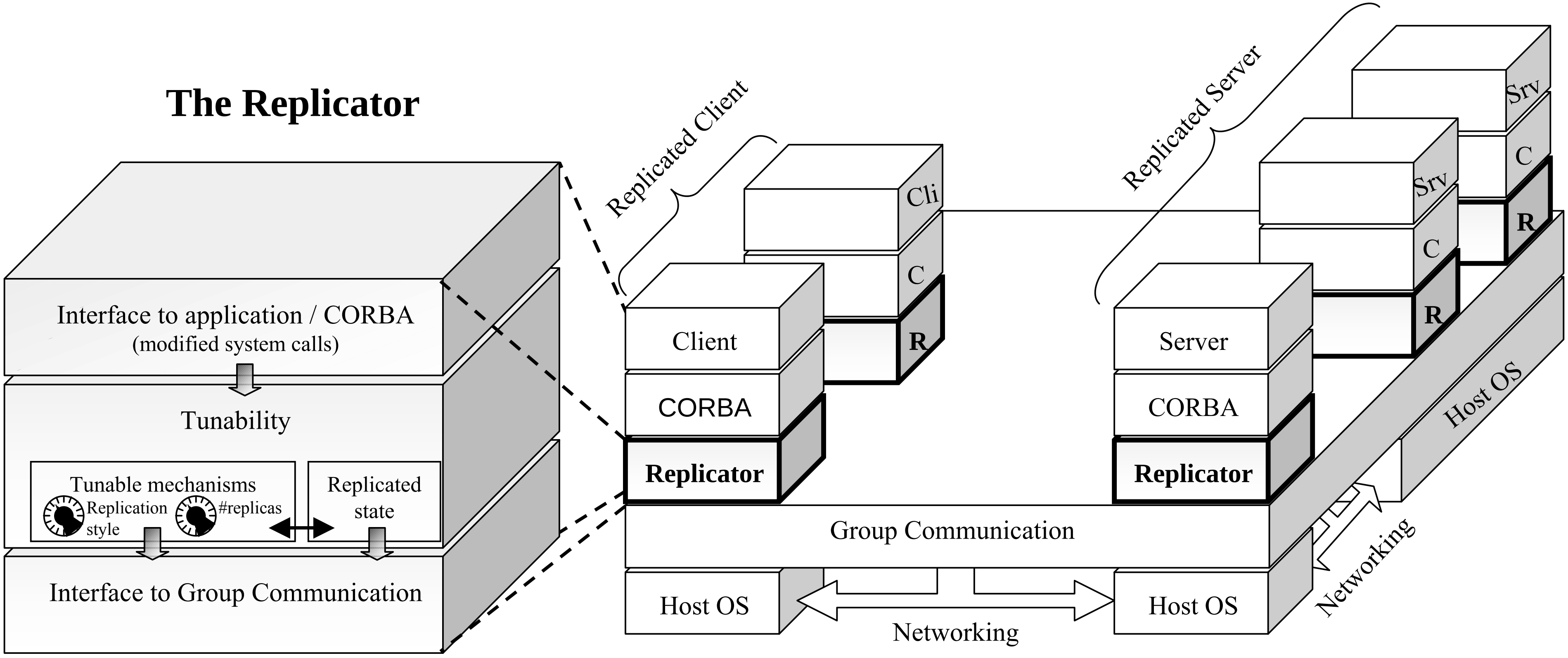
  Adaptive

  Dependability

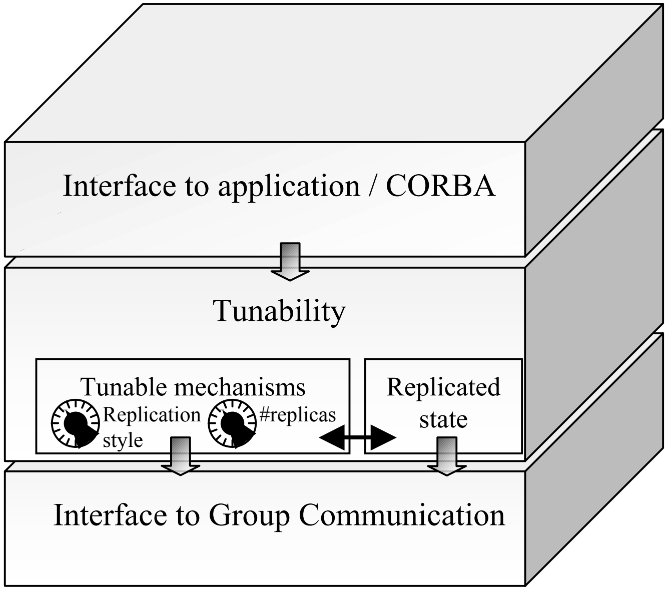
  (www.ece.cmu.edu/~mead)



Architecture for Versatile Dependability



The Replicator



- **Library interposition**
  - Intercepts and redefines system calls
  - Provides transparency without modifying the application, the middleware, or the OS
- **Group membership and communication**
  - The Spread toolkit
- **Replicated state**
  - Decisions made based on information already available at every host
- **Tunable fault-tolerant mechanisms**
  - Replication style, number of replicas, checkpointing style and frequency
  - Represent the low-level knobs
- **Adaptation Policies**
  - Implement the high-level knobs

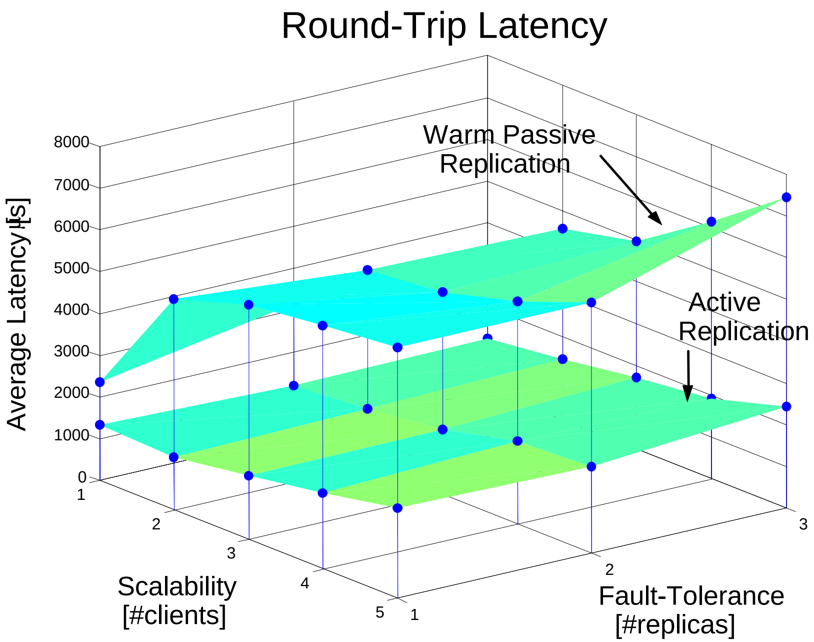
Case Study: Tuning Scalability

Increasing *Scalability*
(accommodating more clients)

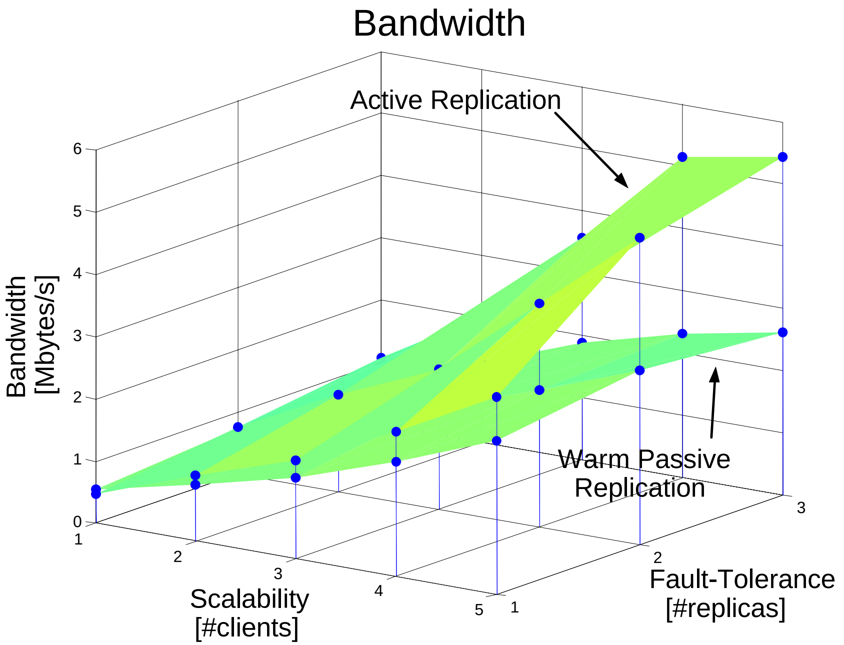
- ☛ **Using more *resources***
  (e.g., CPU, bandwidth)
- ☛ **Decreasing *performance***
  (e.g., response time)
- ☛ **Decreasing *fault-tolerance***
  (redundancy levels)

Exposing System Trade-offs

Comparing active and passive replication in terms of round-trip latency and bandwidth



Round-Trip Latency

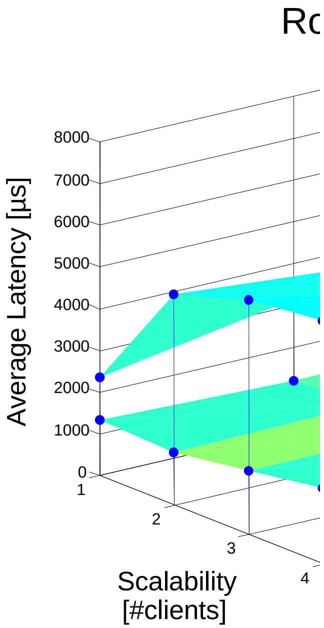


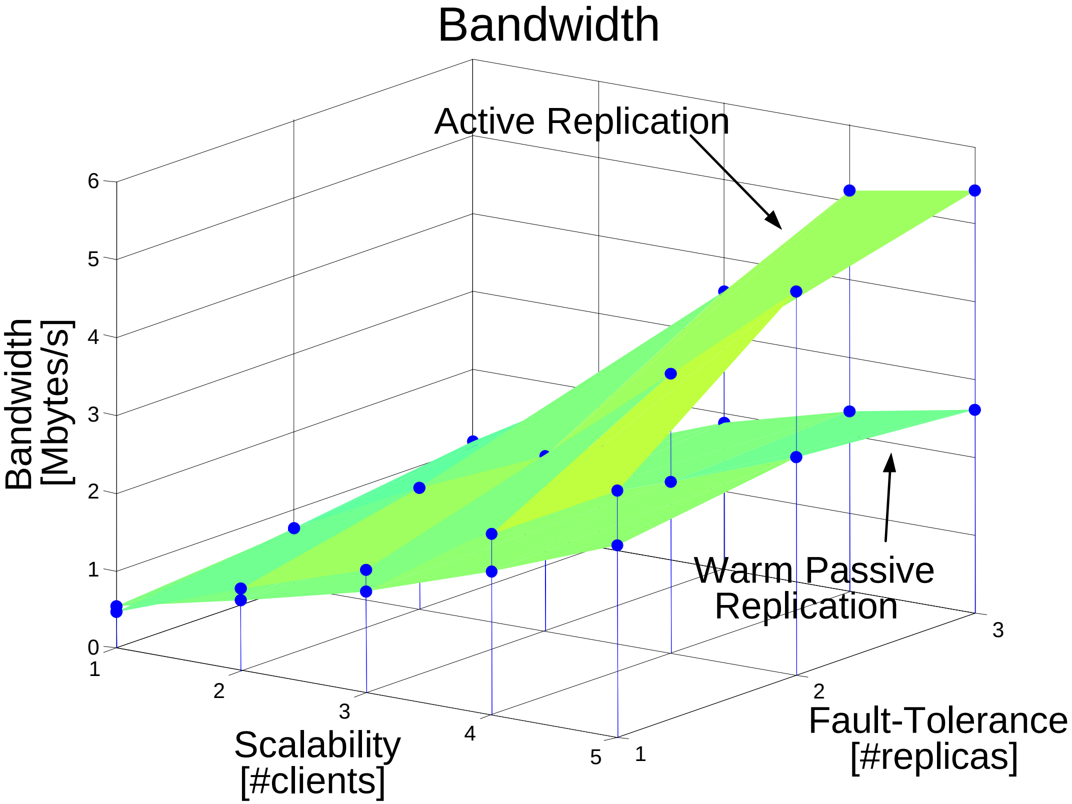
Bandwidth

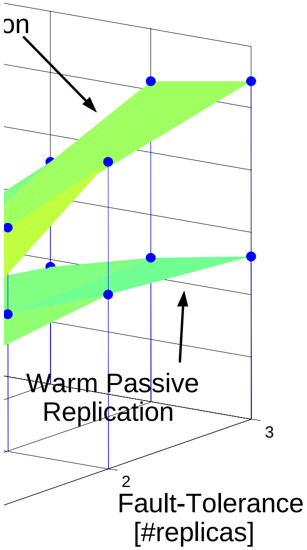
Exposing System Trade-offs

Warm passive replication uses less bandwidth

Active replication has lower latency







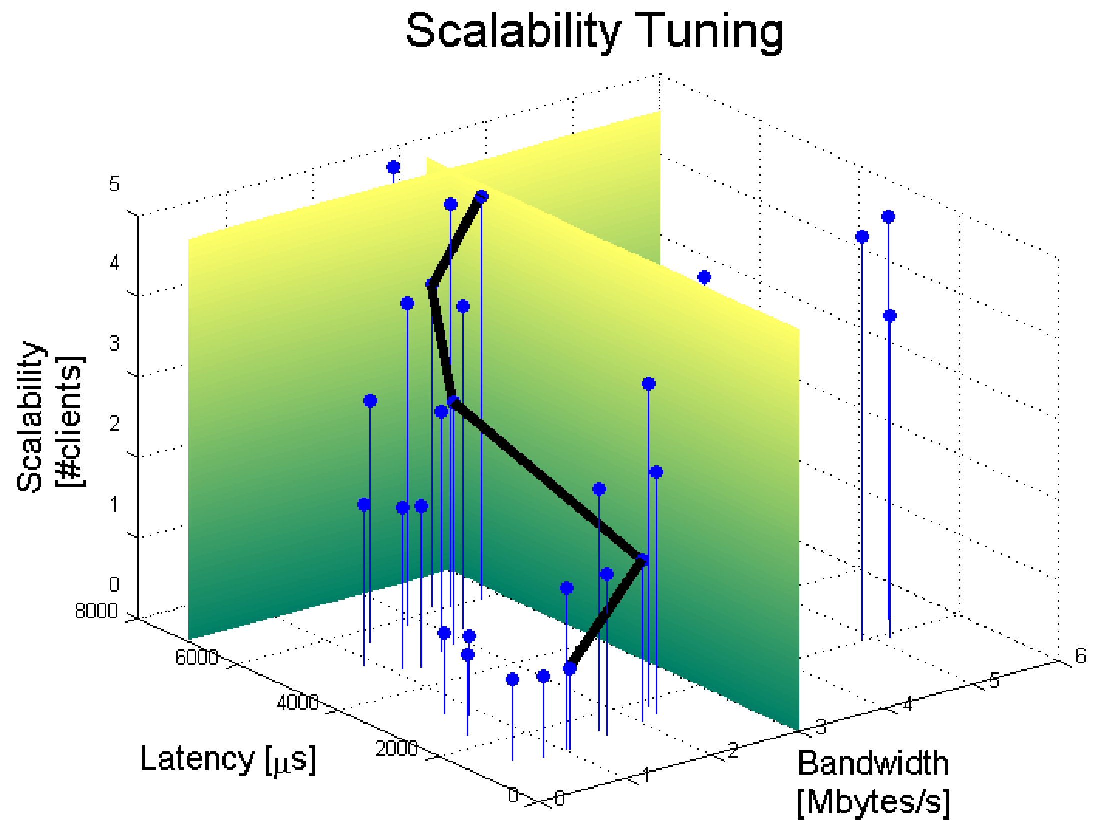
System Constraints

- **Implementing a “scalability” knob under bandwidth, latency and fault-tolerance constraints**

- **Requirements:**
  1. The average latency shall not exceed 7000 µs
  2. The bandwidth shall not exceed 3MB/s
  3. The configuration should tolerate as many crash faults as possible
  4. The following formula should be used to break any ties:

$$\mathrm{Cost}_{\mathrm{i}} = p\frac{Latency_i}{7000\,\mu s} + (1-p)\frac{Bandwidth_i}{3MB/s}$$

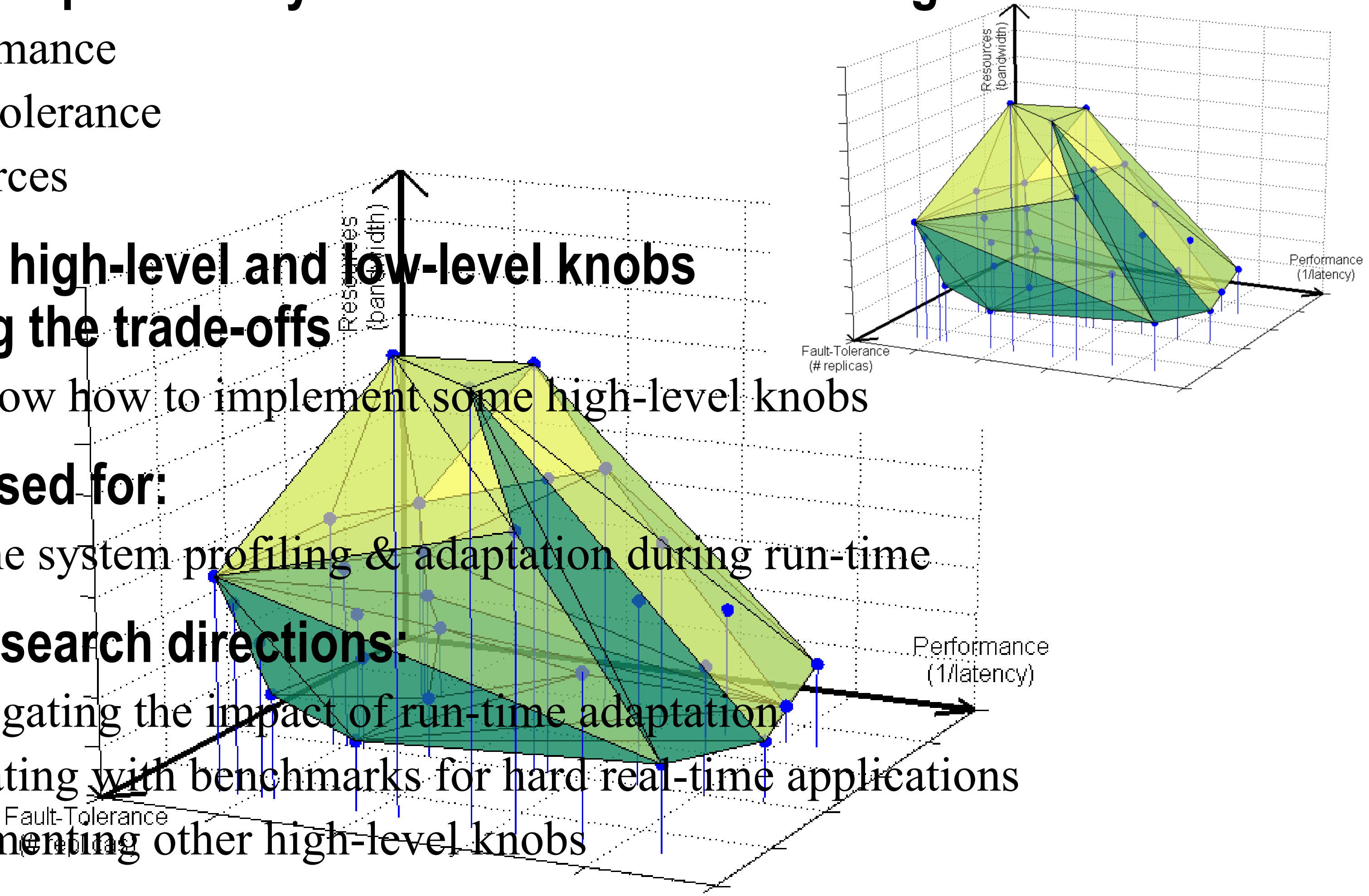
Implementing a “Scalability” Knob



# Clients	1	2	3	4	5
Configuration	Active (3)	Active (3)	Passive (3)	Passive (3)	Passive (2)
Latency	1246 µs	1457 µs	4966 µs	6141 µs	6006 µs
Bandwidth	1.05 MB/s	2.03 MB/s	1.89 MB/s	2.32 MB/s	2.8 MB/s
#Faults tolerated	2	2	2	2	1

Conclusions

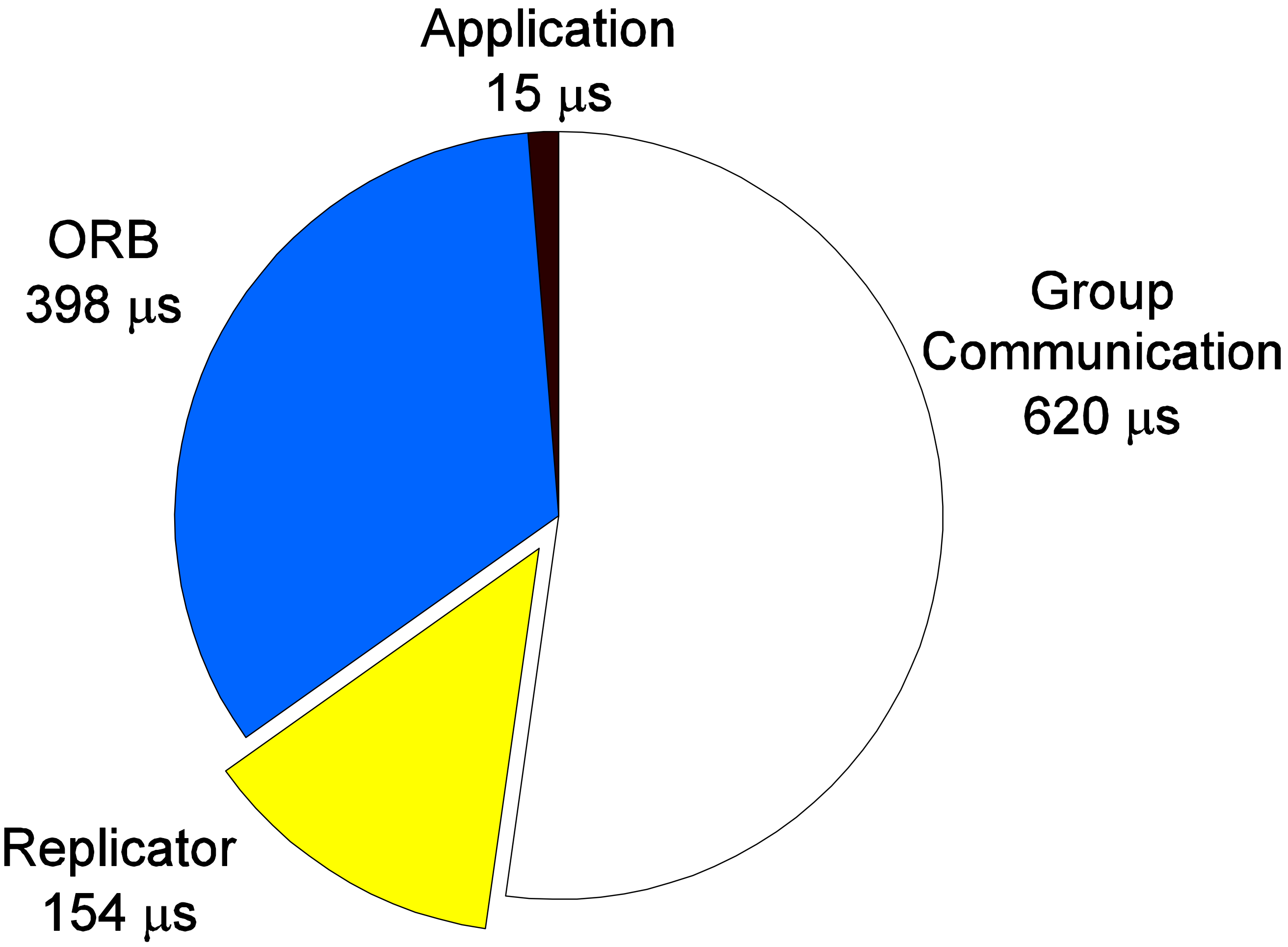
- **Versatile dependability tunes the trade-offs among:**
  - Performance
  - Fault-tolerance
  - Resources
- **Provides high-level and low-level knobs for tuning the trade-offs**
  - We know how to implement some high-level knobs
- **Can be used for:**
  - Off-line system profiling & adaptation during run-time
- **Future research directions:**
  - Investigating the impact of run-time adaptation
  - Evaluating with benchmarks for hard real-time applications
  - Implementing other high-level knobs



Thank You!

For more information: www.ece.cmu.edu/~tdumitra

Performance of the Architecture



For More Information

http://www.ece.cmu.edu/~tdumitra

Tudor Dumitraş
Ph.D. Student
ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Tel: +1-412-268-5005
tdumitra@ece.cmu.edu

Motivation

- **Dependable system architectures currently lack the flexibility to adapt to the operating environment**

- **Behavior of the system depends on static fault assumptions**

- **No generic framework for resolving conflicts among requirements**

- **Architectures should become tunable to provide support for:**
  - Configuring the system before deployment
  - Adapting to changes in the environment during run-time
  - Maintaining the system throughout its life-cycle