Dependability and Adaptivity in Cactus, Self✷, and iMobile: Challenges and Solutions

Matti Hiltunen

AT&T Labs-Research
180 Park Avenue
Florham Park, NJ

hiltunen@research.att.com

- Cactus: Event-based framework for configurable and adaptive distributed services and protocols

- Self✷: Data-flow based component framework for pervasive dependability

- iMobile: Mobile enterprise services platform
  - New name: AMN (AT&T Mobile Network)

Acknowledgements

- Cactus: Joint work with Rick Schlichting (AT&T Labs), Nina Bhatti (currently at HP Labs), Patrick Bridges (current at the Univ. of New Mexico) and other former and current graduate students at the Univ. of Arizona
- Self✷: Work by Karin Högtedt and Christof Fetzer at AT&T Labs
- iMobile: Joint work with Robin Chen, Rittwik Jana, etc. at AT&T Labs

AT&T

Outline

Definitions

Example systems: Cactus, iMobile, Self✸

Issues in

- Dependability
- Adaptivity

Dependability and adaptivity in Cactus, iMobile, Self✸

Conclusions

Definitions

Middleware:

- If it is not an application and it is not part of the operating system, it must be middleware
- Software layers/components that provide higher level abstractions for application designers
- Middleware must typically provide support for more than one application (i.e., *generality*)

Customizable middleware can be tailored to provide different service properties/attributes to different applications.

Adaptive middleware can change its behavior at runtime as a reaction to changes in the execution environment or application requirements.

Dependable middleware is

- itself dependable
- increases the dependability of the applications that use the middleware

Outline

Definitions

Example systems: Cactus, iMobile, Self✸

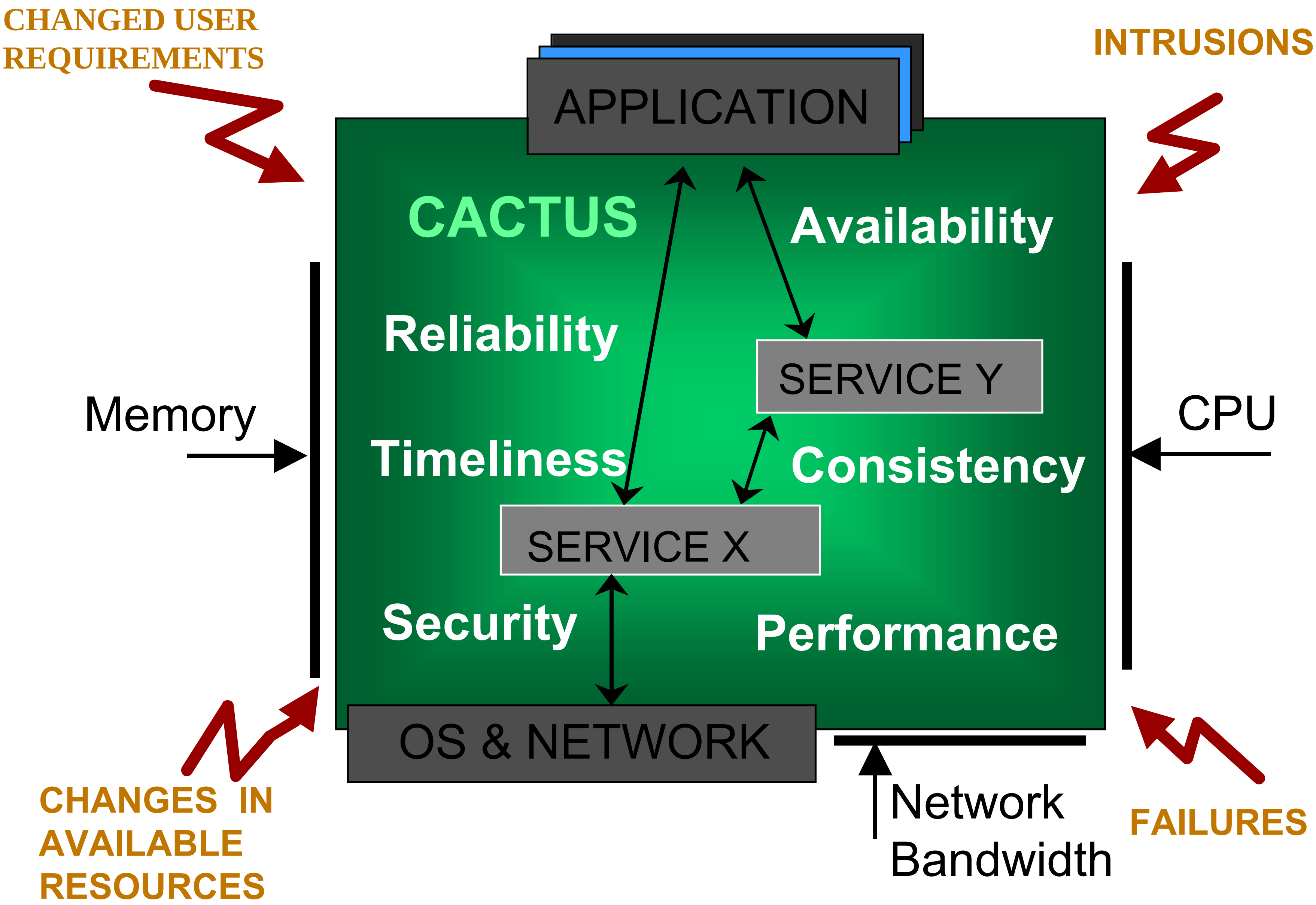
Issues in

- Dependability
- Adaptivity

Dependability and adaptivity in Cactus, iMobile, Self✸

Conclusions

Cactus Vision



Cactus Approach

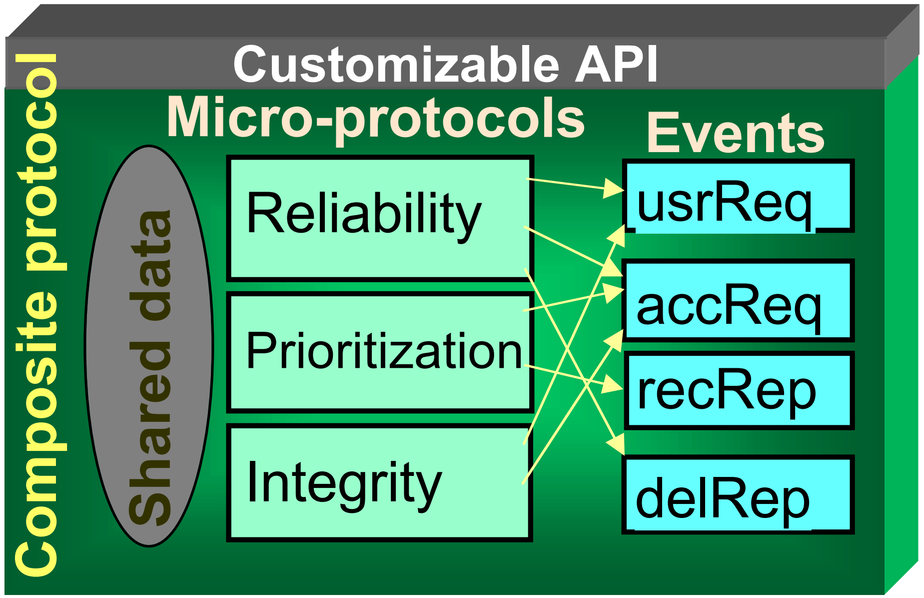
Cactus goals:

- Configurable services with highly-customizable functionality and properties.
- Dynamically adaptive services that can change their behavior and properties at runtime.

Service = a composite protocol consisting of micro-protocols, each of which implements a function or property.

Configuration = choose micro-protocols.

Adaptation = activate/deactivate micro-protocols at runtime.



Micro-protocol execution:

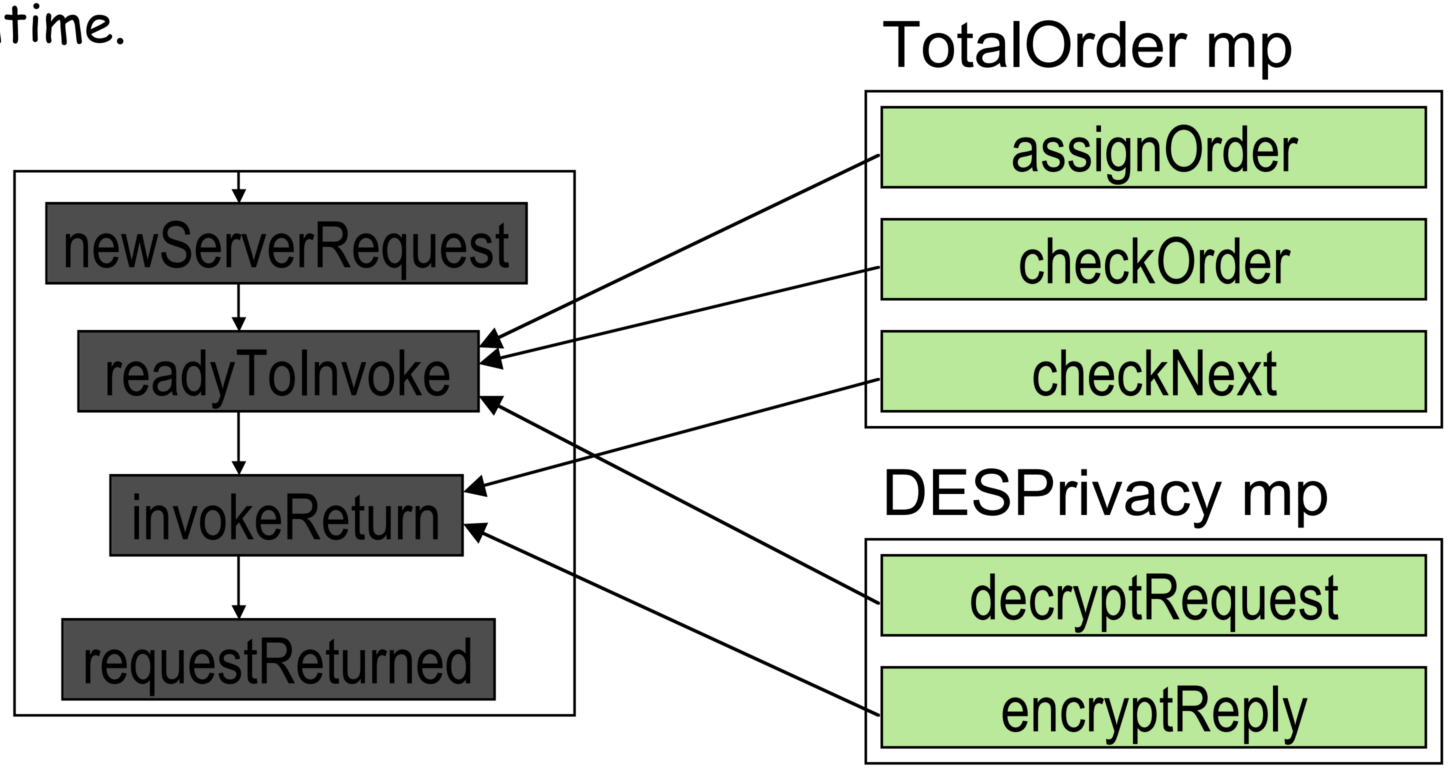
Micro-protocol = collection of event handlers.

Handlers bound to events in the composite protocol.

- Binding dynamic, can be changed at runtime.
- Order can be specified (often important).

Events provide a level of indirection between micro-protocols.

Allow new micro-protocols to be loaded and activated at runtime.



Cactus Prototypes and Services

Prototypes:

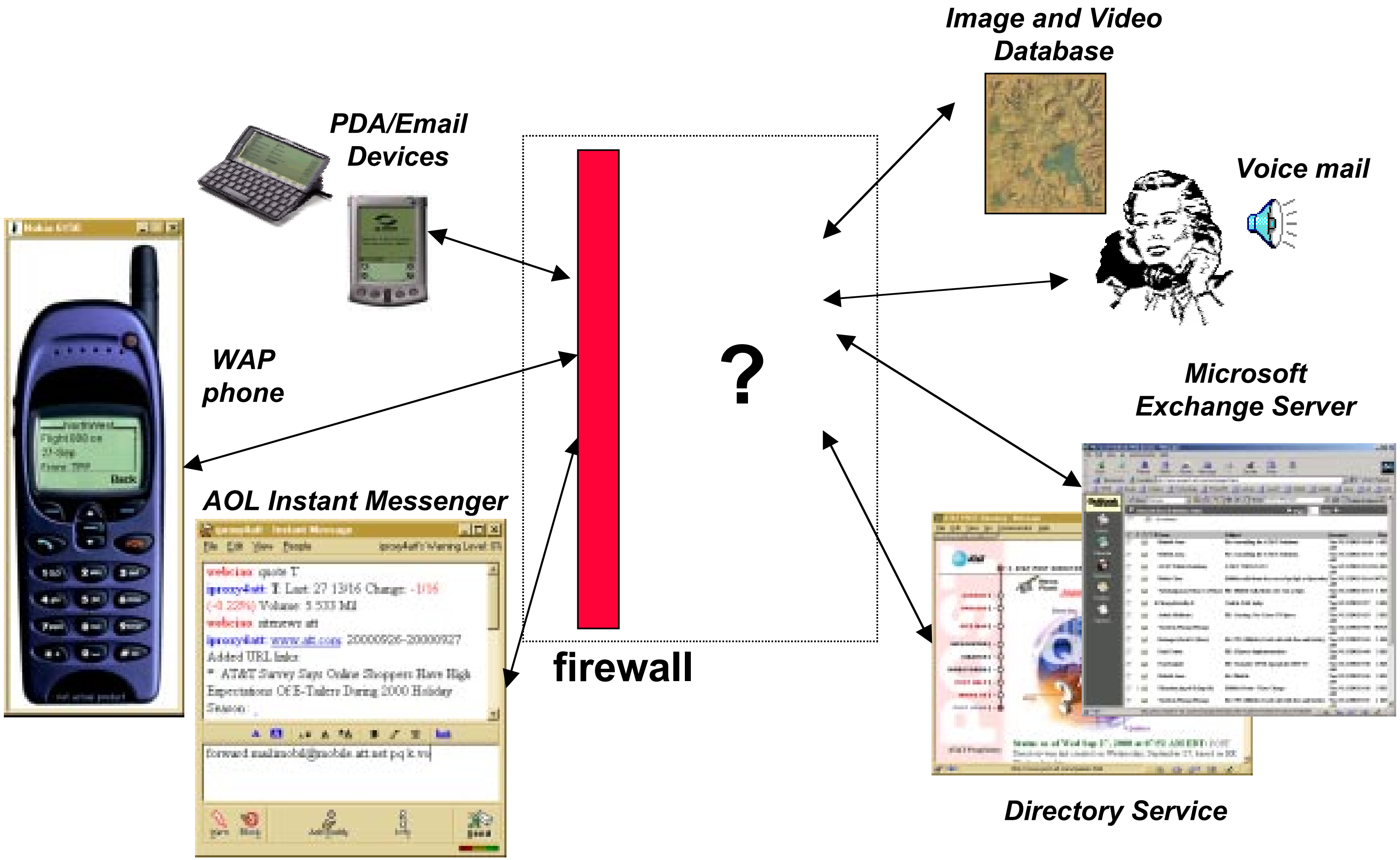
- Cactus/C on Linux, Cactus/C++ on Linux and Solaris, and Cactus/Java.

Example configurable services:

- Fault-tolerance services: group RPC, membership, system monitoring.
- Real-time services: RTD channels (communication).
- Secure communication services: SecComm.
- Services that address multiple QoS attributes: CQoS

Services range from transport protocols to middleware and application services.

iMobile: Mobile Enterprise Services

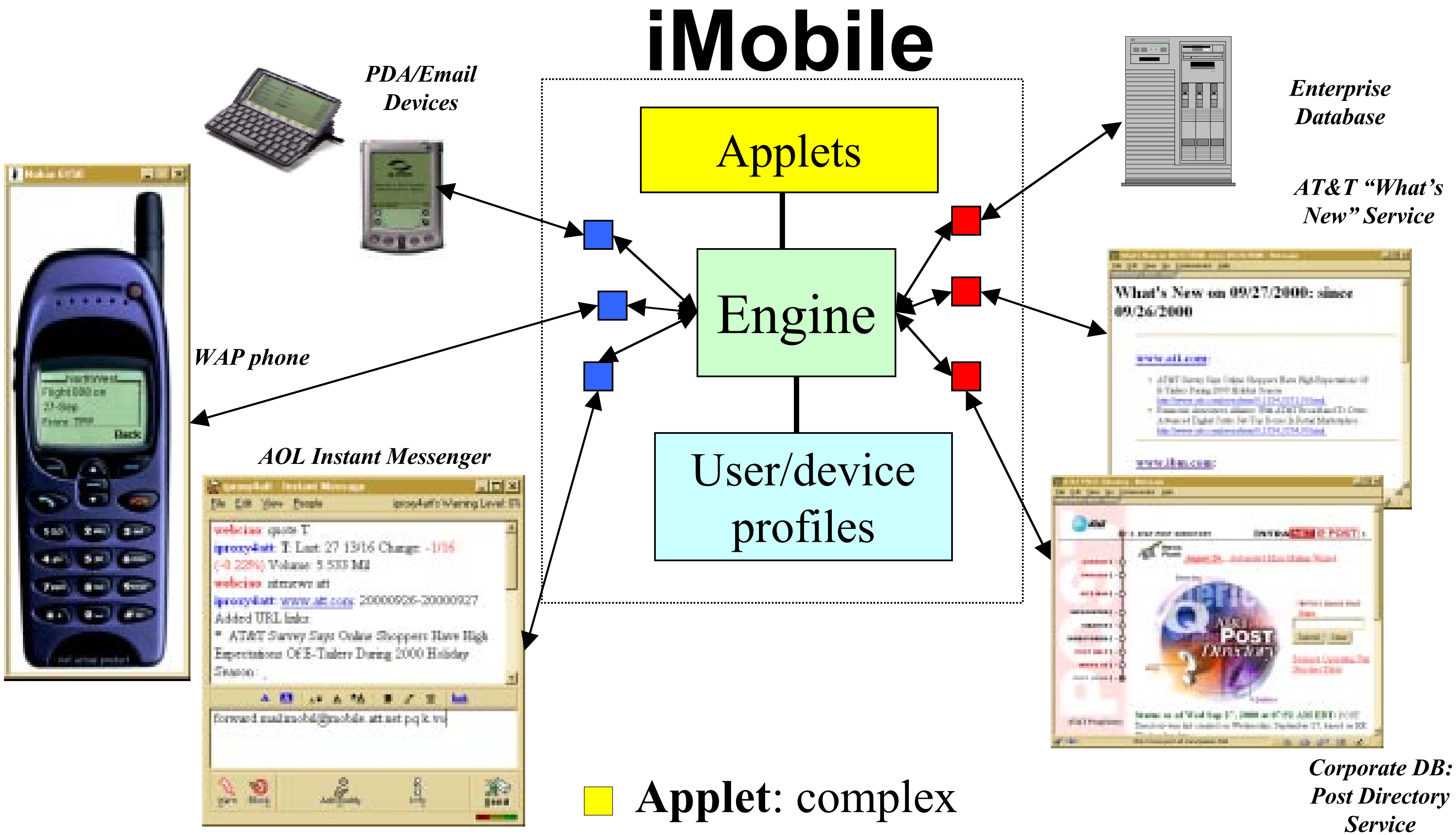


Devices

Info Space

What is missing?

iMobile: Logical Architectural View

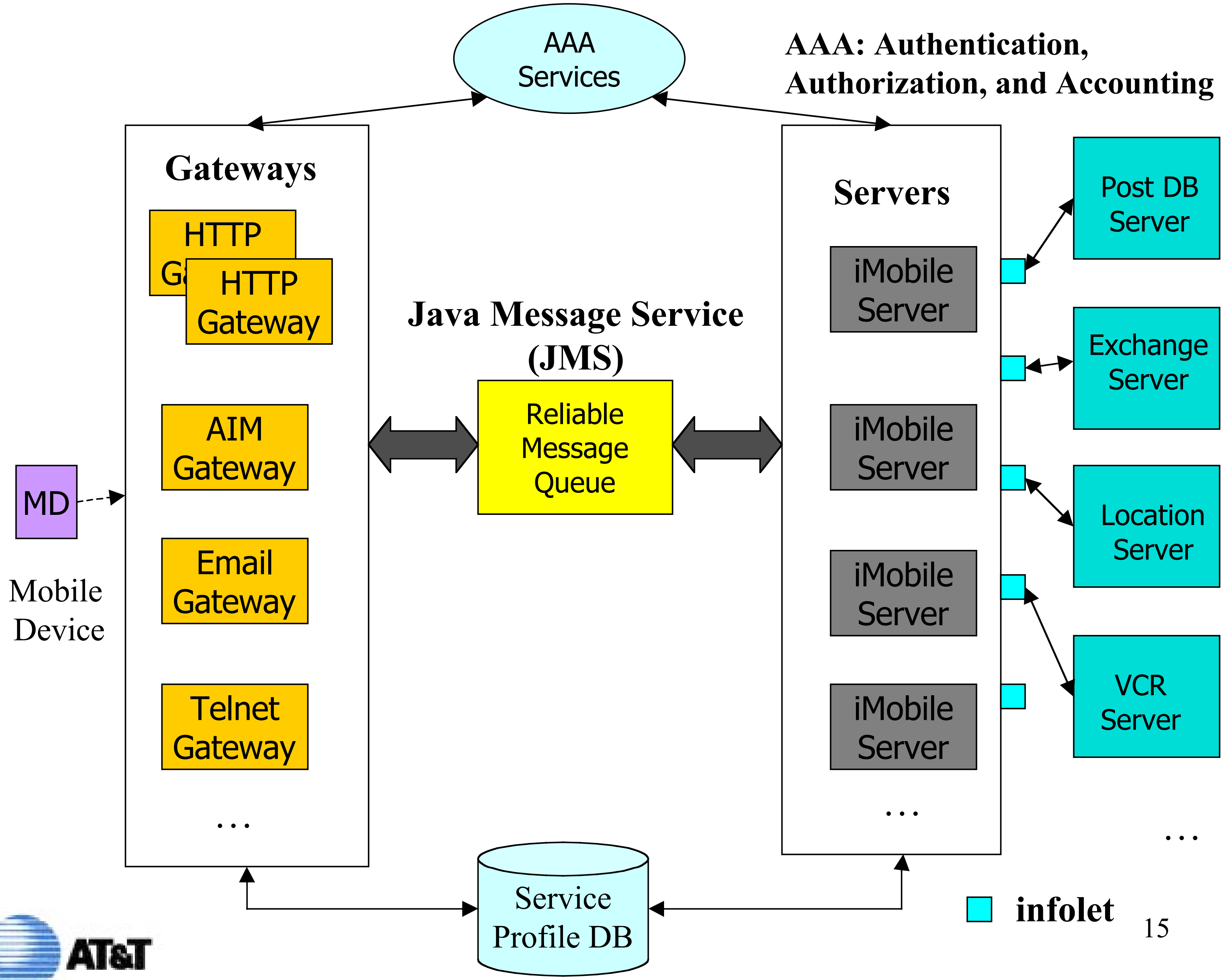


- **Devlet**: device driver
- **Applet**: complex application logic
- **Infolet**: interface to an info space

iMobile Access to Critical Enterprise Applications

- *Authorized users can access:*
  - *Messaging services*
  - *Corporate directories and databases*
  - *Exchange Services: calendar, contacts, inbox, etc.*
  - *Images/Engineering Drawings*
  - *Instructional Videos*
  - *Voice Mail*
  - *…*

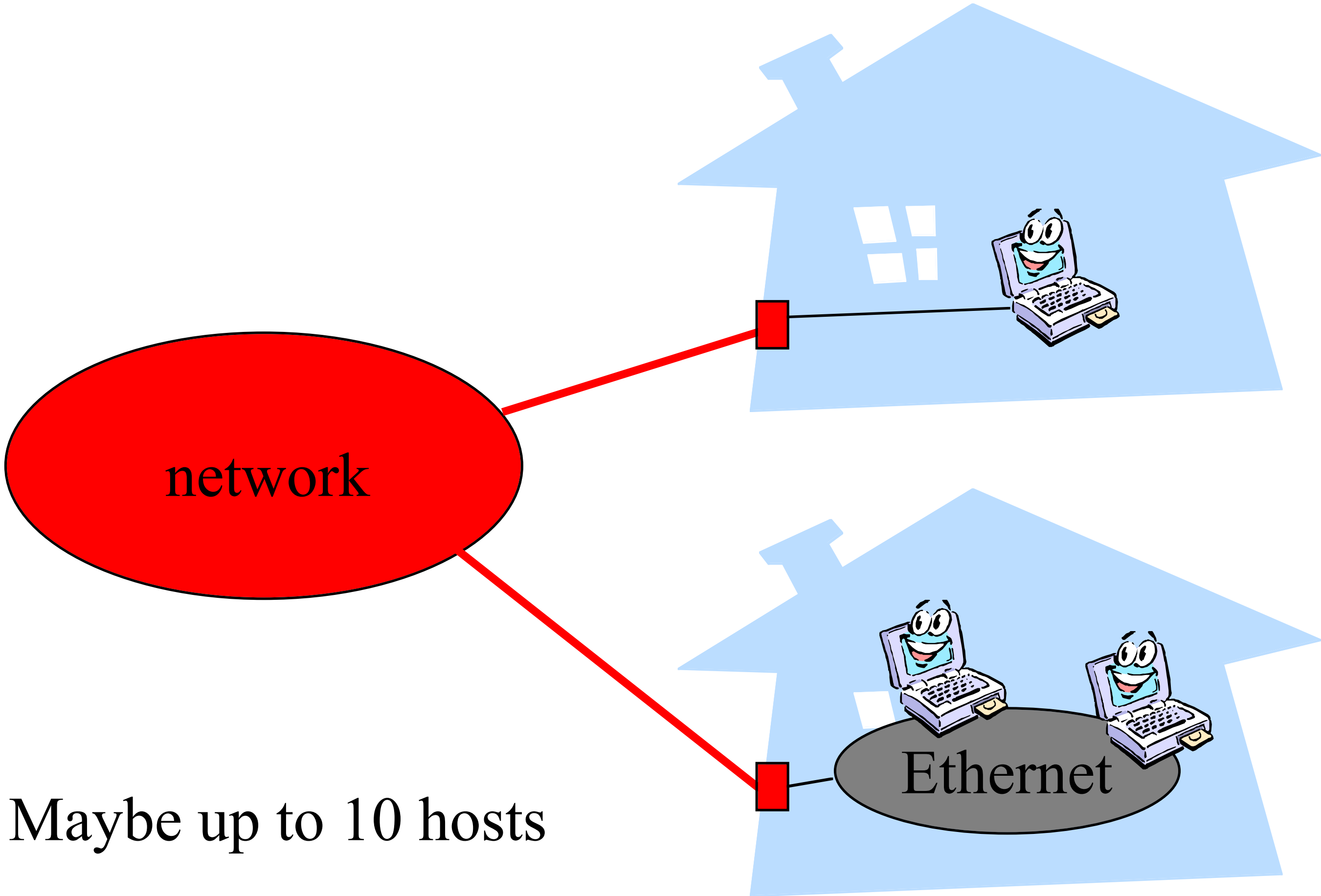
iMobile Enterprise Edition



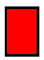
Standards Compliant and Carrier Agnostic

- Standard Enterprise Software
  - Message Oriented Middleware: JMS
  - Corporate Database: JDBC
  - Corporate Directory: LDAP, JNDI
  - Microsoft Exchange Server: WebDAV
  - Enterprise VPN Products: IPSec
  - Content Transcoding: XML, XSLT
  - Messaging Services: SMTP, IMAP, POP3

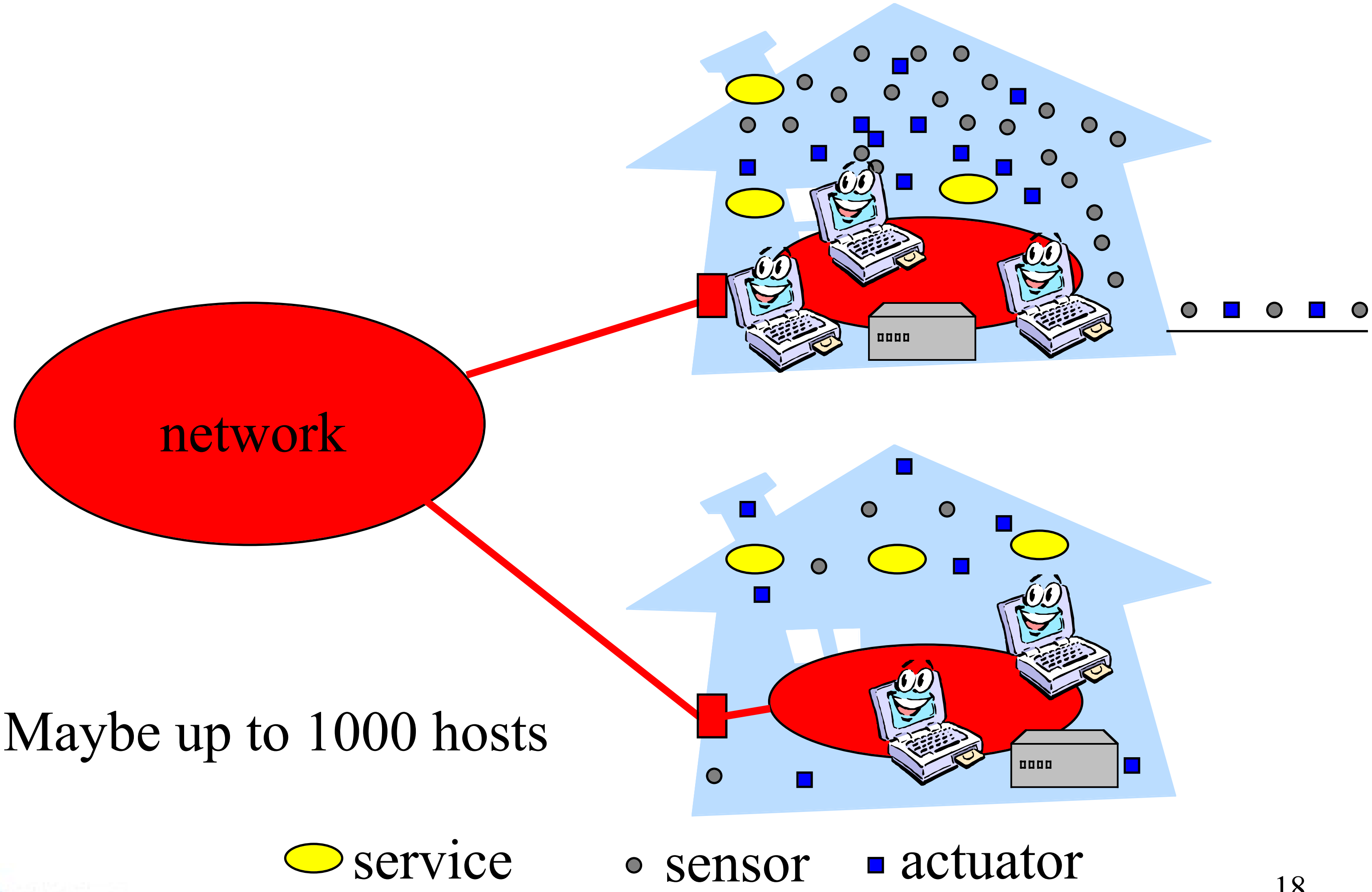
Self✸
Motivation: Home Networks



Maybe up to 10 hosts


dsl/cable modem

Future: More Nodes & Services



Maybe up to 1000 hosts

Pervasive Dependability

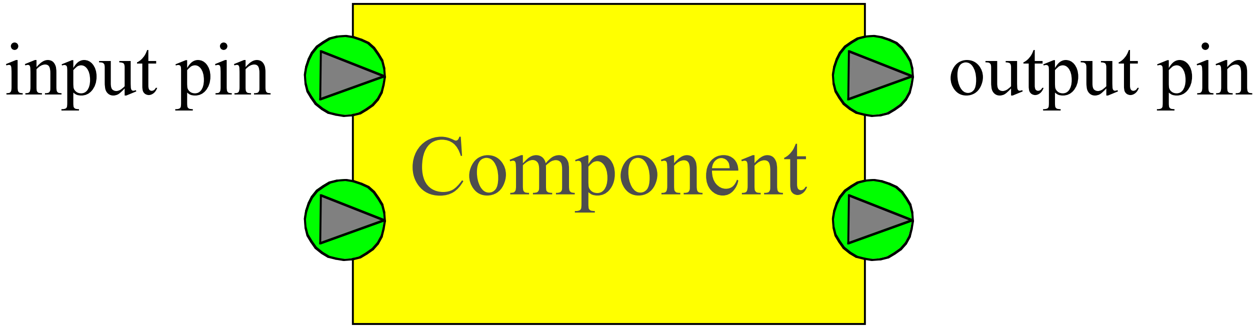
- Pervasive systems are society critical
- System management has to be cheap
    - customers might pay low monthly fee
    - each customer service call costs money
- Automatic (=cheap) system management is needed to make pervasive systems a reality

Research issues

- *self-management*
- *self-diagnosis,*
- *self-customization,*
- *self-configuration,*
- *self-reconfiguration,*
- *self-**

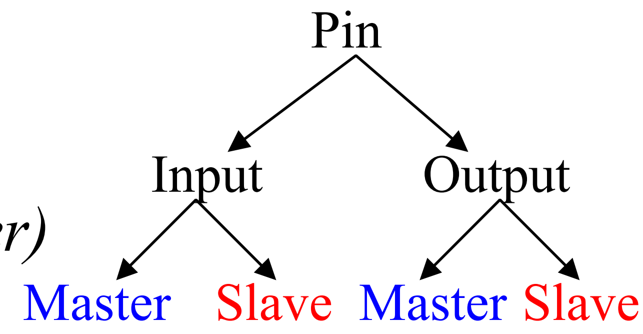
⅄ *Approach: Developing Self ✸, a dataflow-oriented framework to use as test-bed*

Components & Pins



- *Components have any number of pins*
- *Pins can only be connected pair-wise*
- *Pins are unidirectional (input or output)*
- *Pins are untyped (i.e., accept any object)*
- *All communication via pins*

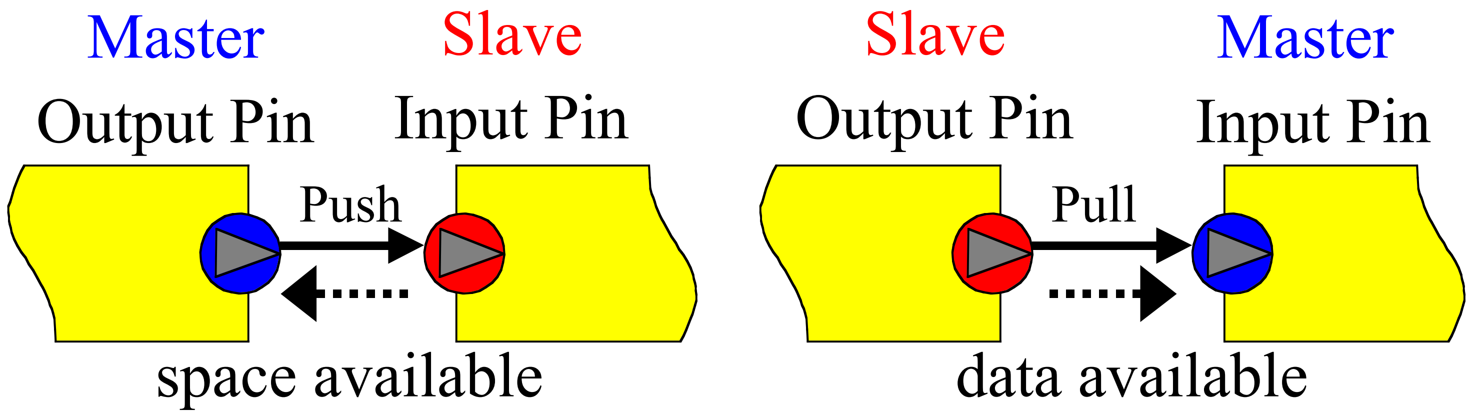
Master and Slave Pins



Connection – 2 pins together

1 Master pin provides control (caller)

1 Slave pin is controlled (callee)

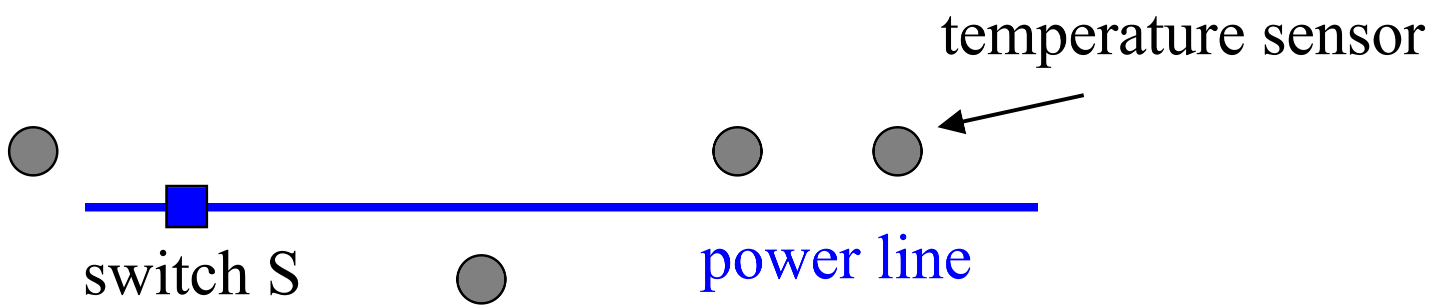


- Both blocking and non-blocking functionality
- Standard adapter components used to fix pin mismatches

Self✸ Toolkit

- *Self✸ Library*
  - *Standard Components*
  - *Standard Pins*
- *User defined*
  - *Components*
  - *Pins*
- *Generators*
  - *xml2C*

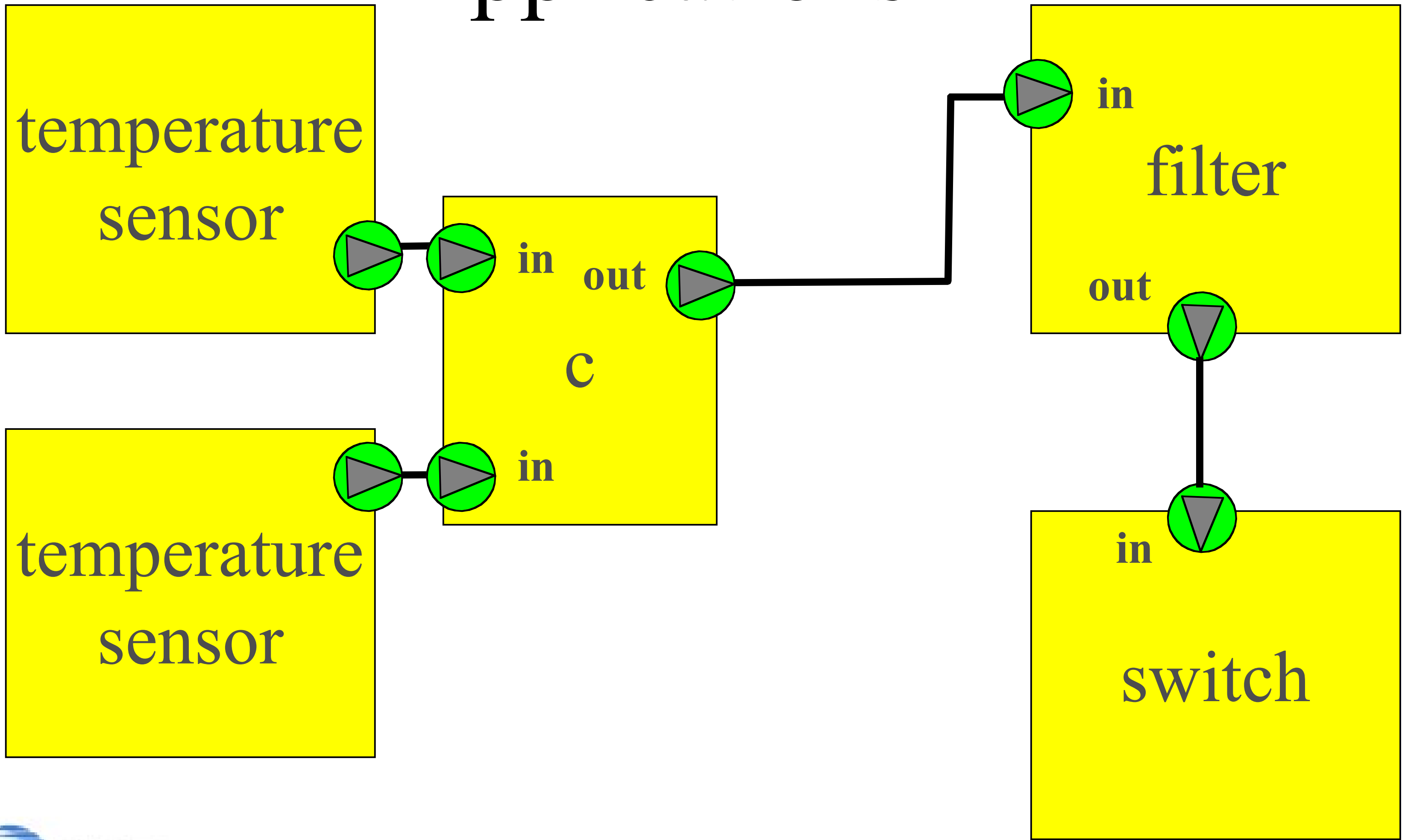
Example Application



if (Temperature > Threshold)
switch off S

Within 1sec, 99.999%

Dataflow-Oriented Applications



Outline

Definitions

Example systems: Cactus, iMobile, Self✸

Issues in

- Dependability
- Adaptivity

Dependability and adaptivity in Cactus, iMobile, Self✸

Conclusions

Issues in Dependability

Middleware CAN increase the dependability of applications (e.g., Isis, FRIENDS, Eternal, QuO/AQuA,...).

... but ...

it MAY also decrease the dependability of applications.

Middleware introduces additional software components - and hardware components, e.g., key distribution server or replication manager - that may fail/ become security vulnerability/etc.

Solutions: no magic bullet

- Use of dependability mechanisms: replication, encryption, failure detection, etc.
- Measurement and analysis to determine the necessary level of redundancy etc.

Issues in Adaptivity

Adaptivity can improve the dependability of the middleware as well as the applications using it.

Adaptation mechanisms:

- Value adaptation: change execution parameters
- Algorithmic adaptation: change algorithms used
- Resource reallocation: reassign resources based on new operating conditions

Types of adaptation:

- Property preserving
- Property changing (e.g., graceful degradation)

Challenges:

- Policies (when and how)
- Coordination between different adaptive components on a host (i.e., on the different levels) – inter-component coordination
- Optimization
- Coordination between different hosts – inter-host coordination
- Stability
- Overhead of the adaptation mechanisms

Outline

Definitions

Example systems: Cactus, iMobile, Self✸

Issues in

- Dependability
- Adaptivity

Dependability and adaptivity in Self✸, iMobile, and Cactus

Conclusions

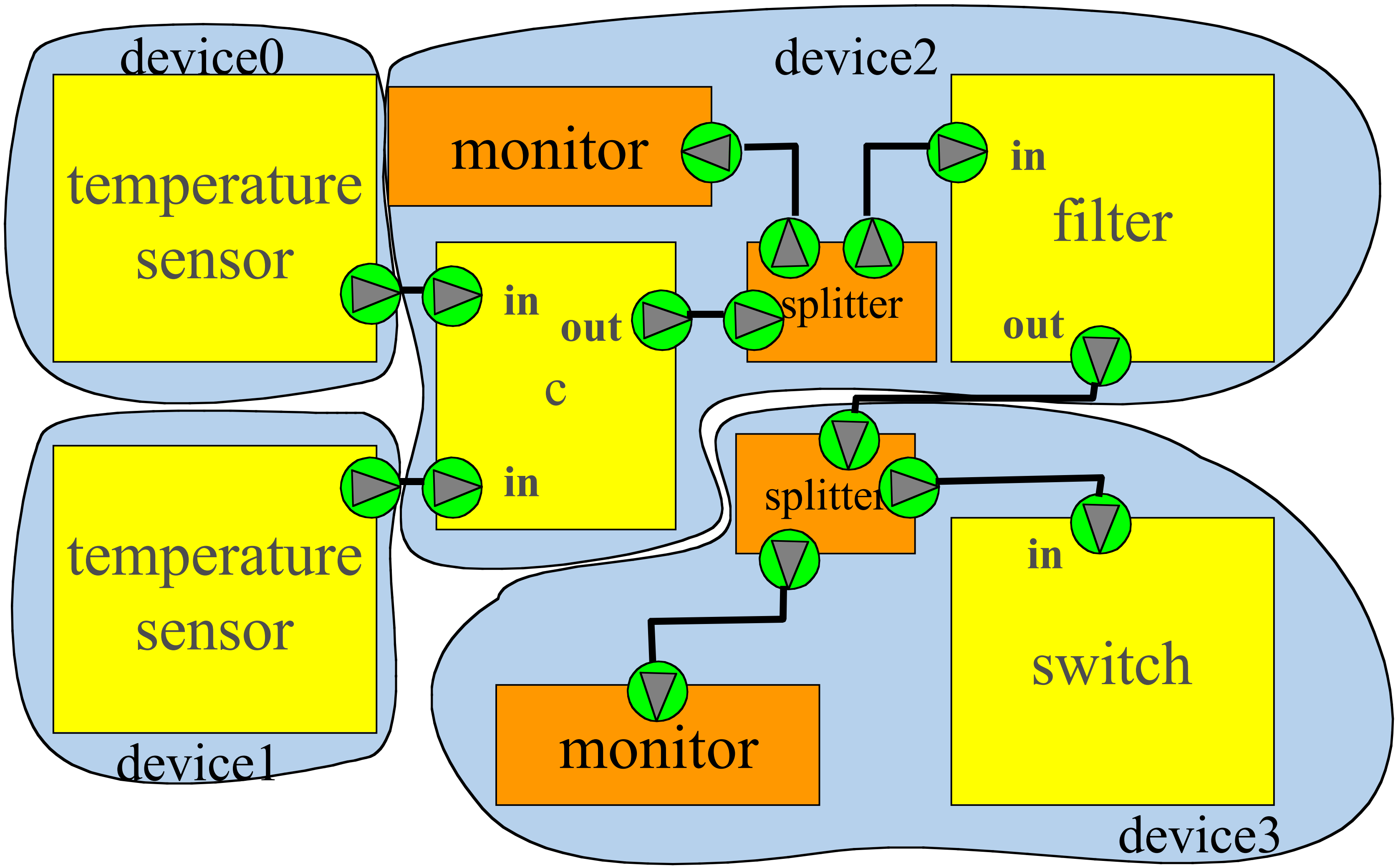
Dependability and adaptivity in Self✷

Dependability mechanisms such as retransmission, redundant transmission along separate paths can be implemented as reusable components

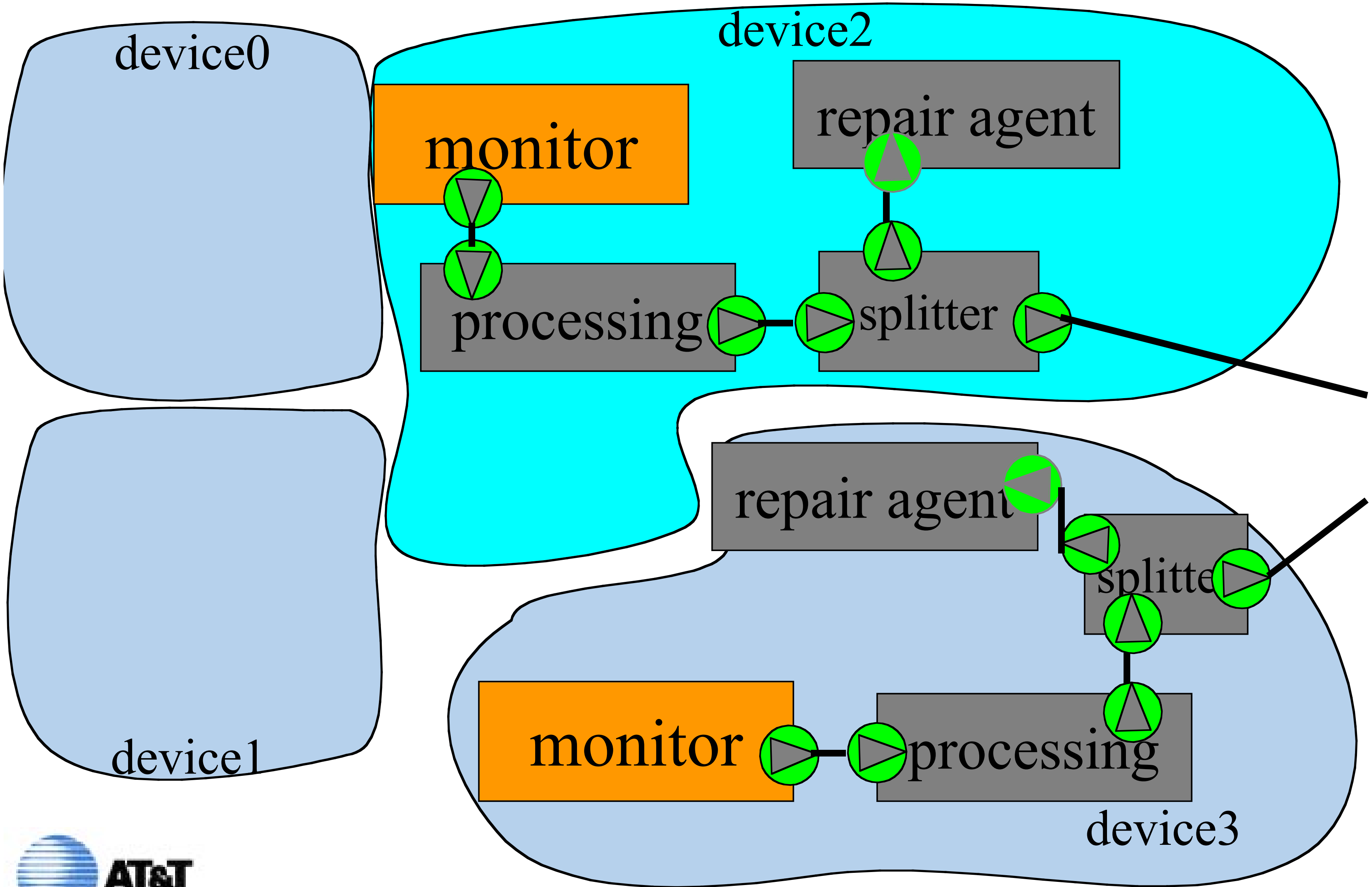
Components themselves - or additional monitoring components - can detect failures and reconfigure the component graph

Component “wrappers” check the component failures

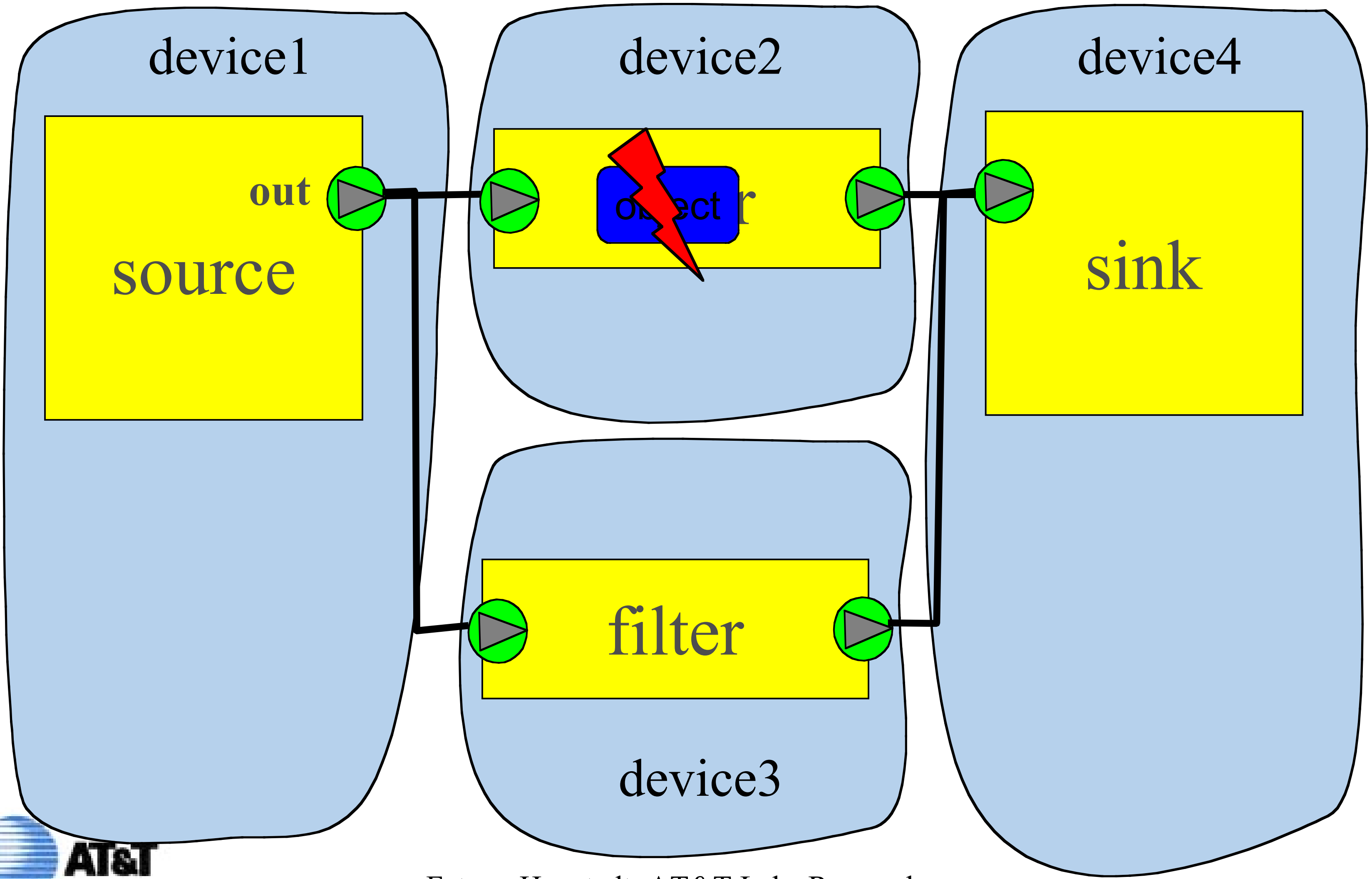
Data Collection



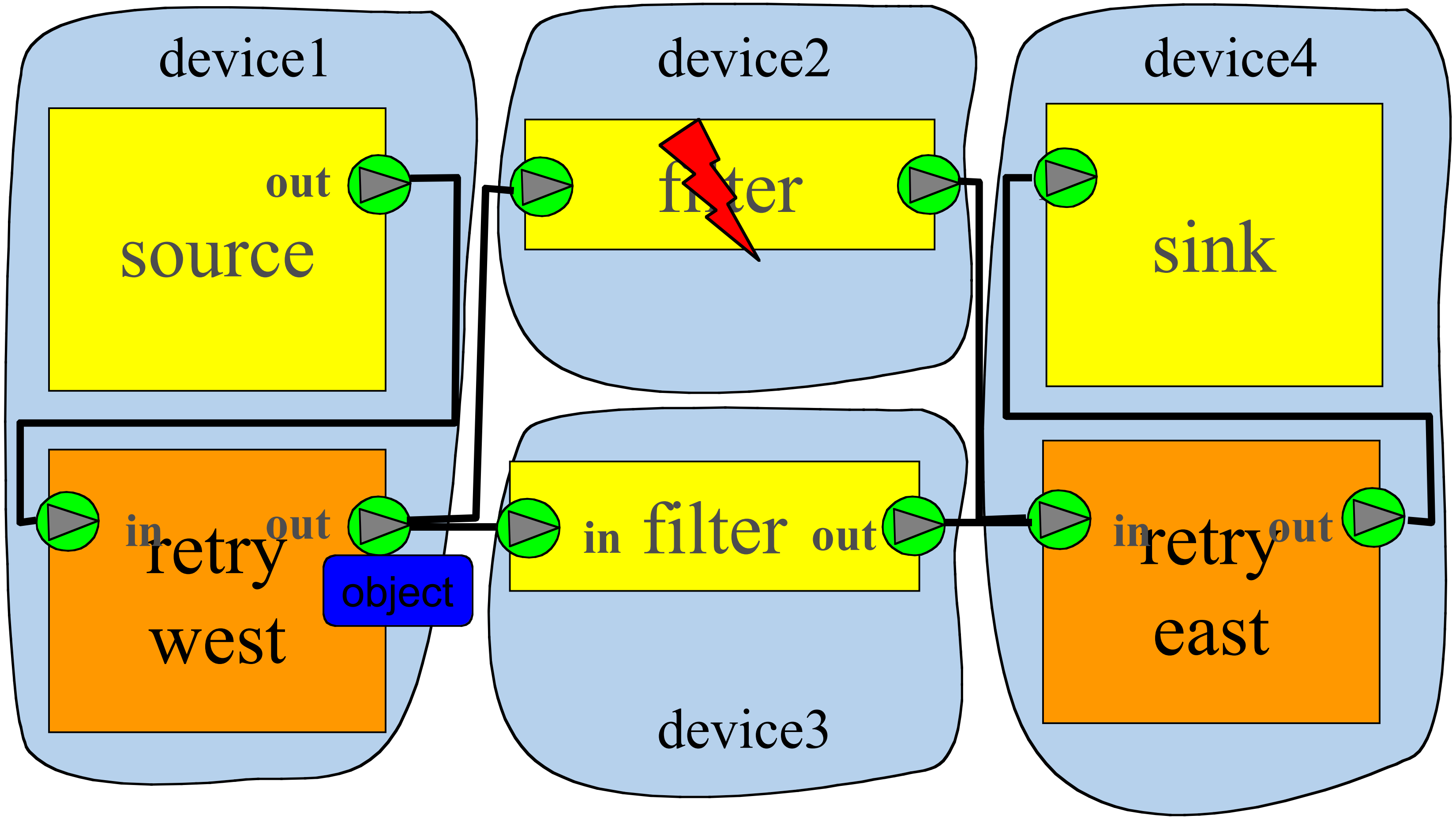
Distributed Data Processing



Self Re-/Configuration Problem



Omission Failure Problem



Dependability in iMobile

Challenge: LOTS of system components:

- gateways, iMobile server, email servers, databases, JMS servers, authentication servers,
- external servers/services accessed by infolets
- wireless connectivity providers (cell phone, paging network, WiFi, etc)

Solutions:

- iMobile servers and gateways redundant
    - Need to add “level 4 redirectors”
- Databases and JMS servers COTS components
  with industry standard interfaces
  $\Rightarrow$ rely on fault-tolerant versions from industry
  (e.g., Oracle database clusters, IBM MQ)
- Component monitoring and automatic restart
- Retransmission of requests.

Solutions (cont.):

- Performance measurement and failure detection (syslog-based running, SNMP planned)
- Multiple access devices/protocols provide redundancy in case one or more wireless access networks is down (SMS vs. Blackberry)

Planned:

- Backup infolets
- Transactional execution semantics
- Use of Java 2 EE facilities
- Cactus in iMobile
- …

Adaptivity in iMobile

Some failure recovery “adaptations” in place.

Planned:

- Dynamic resource allocation based on load/failures
- Algorithmic and value adaptations to deal with high system load and/or component failures (service differentiation, traffic shaping, filtering, etc.)
- Predictive adaptation based on system modeling

Dependability in Cactus

The Cactus framework can be used to implement any dependability mechanisms in a configurable manner.

Example: CQoS.

- a configurable portable QoS architecture for distributed object computing.

CQoS Motivation

Distributed object platforms lack unified support for QoS (fault tolerance, security, and timeliness)

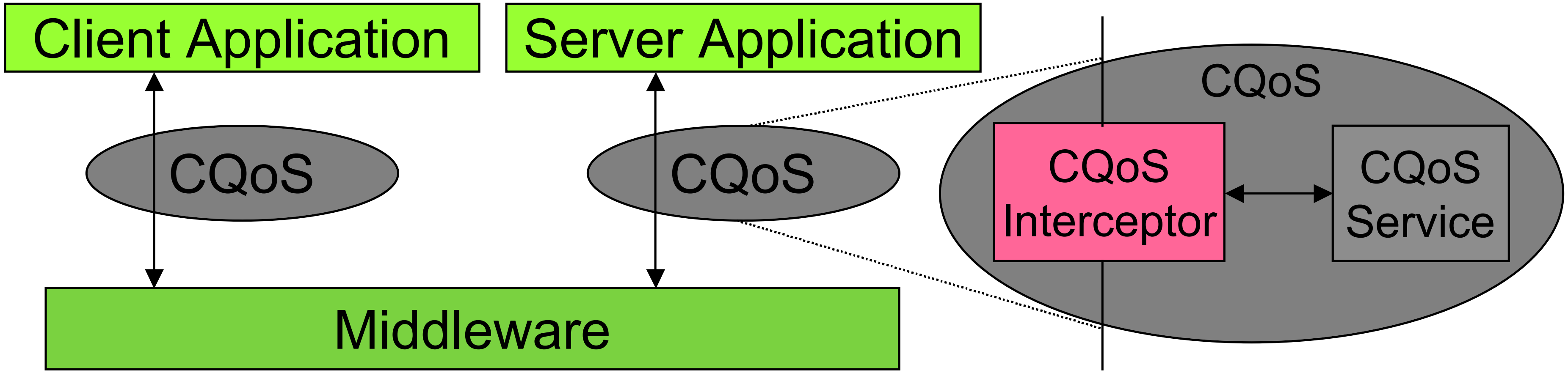
Key observation:

- The fundamental techniques for implementing these QoS attributes are often similar regardless of the specific middleware platform.

Goals:

1. Support highly configurable multi-dimensional QoS with support for fault tolerance, security, service differentiation, and any combination.
2. Platform independent and easily portable to new platforms.

Software Architecture



Client and server applications and the middleware platform unmodified.

CQoS consists of two components:

- Application and platform-specific CQoS interceptor generated from IDL.
- Generic CQoS service component implements QoS.

Separates QoS implementation from specifics of the platform.

Realizing QoS Enhancements

Micro-protocols can be used to implement any function or property

Micro-protocols include:

- Fault tolerance: ActiveRep, PassiveRep, TotalOrder, MajorityVote, … .
- Security: DESPrivacy, …
- Timeliness: PrioritySched, QueueSched, TimedSched.

Different combinations of micro-protocols provide semantically different custom variations of CQoS.

Implementation

A prototype of CQoS has been completed using Cactus/J.

CQoS Interceptors have been implemented for CORBA and Java RMI.

- CORBA: Replace standard stub and skeleton.
- Java RMI: Replace stub, introduce proxy server.
- The generation of CQoS Interceptors has been automated (so far for CORBA).

The CQoS Service components are independent of CORBA/Java RMI - operate on both.

Similar architecture planned for iMobile.

Adaptivity in Cactus

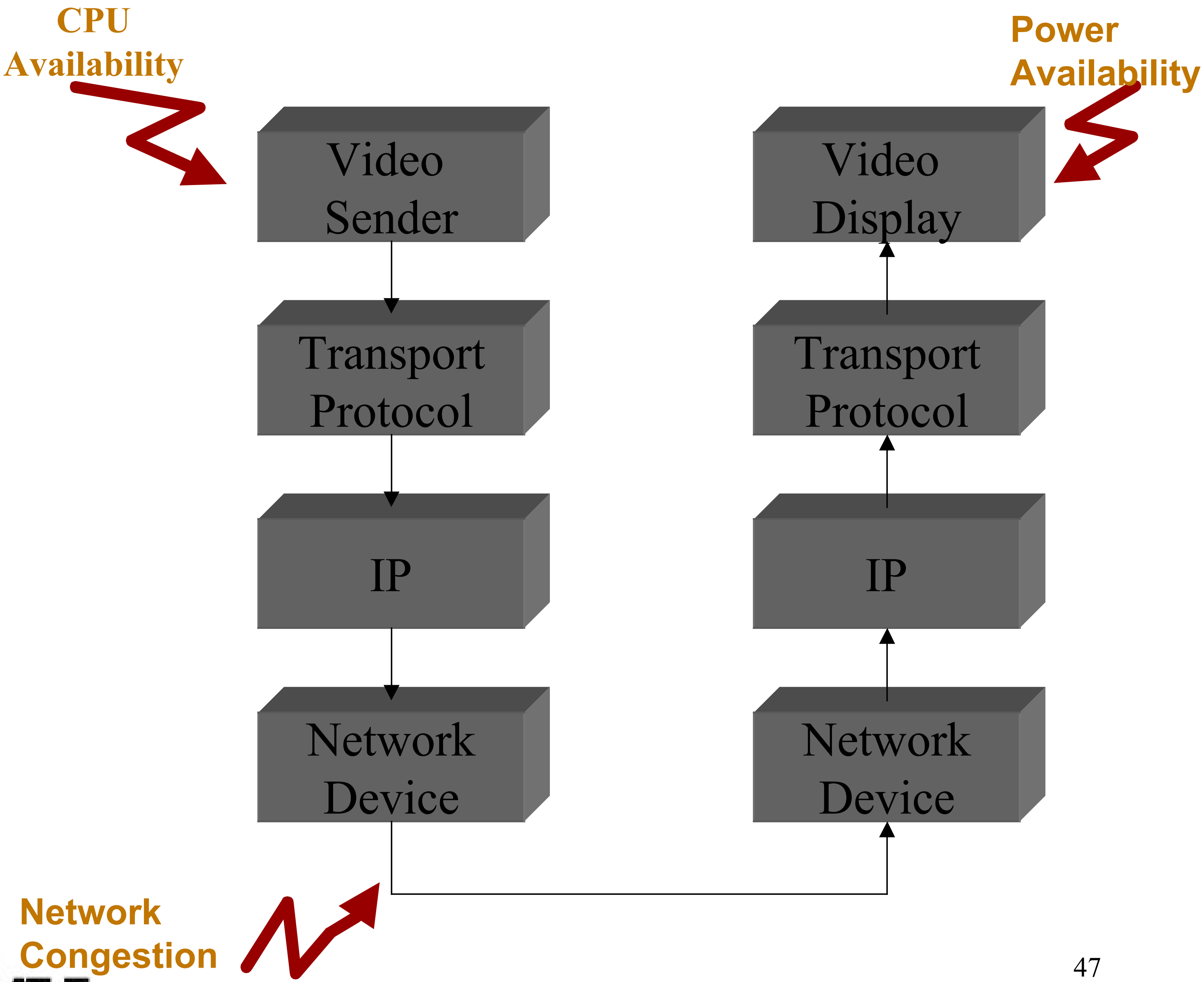
Event mechanism makes it easy to activate and deactivate micro-protocols at runtime.

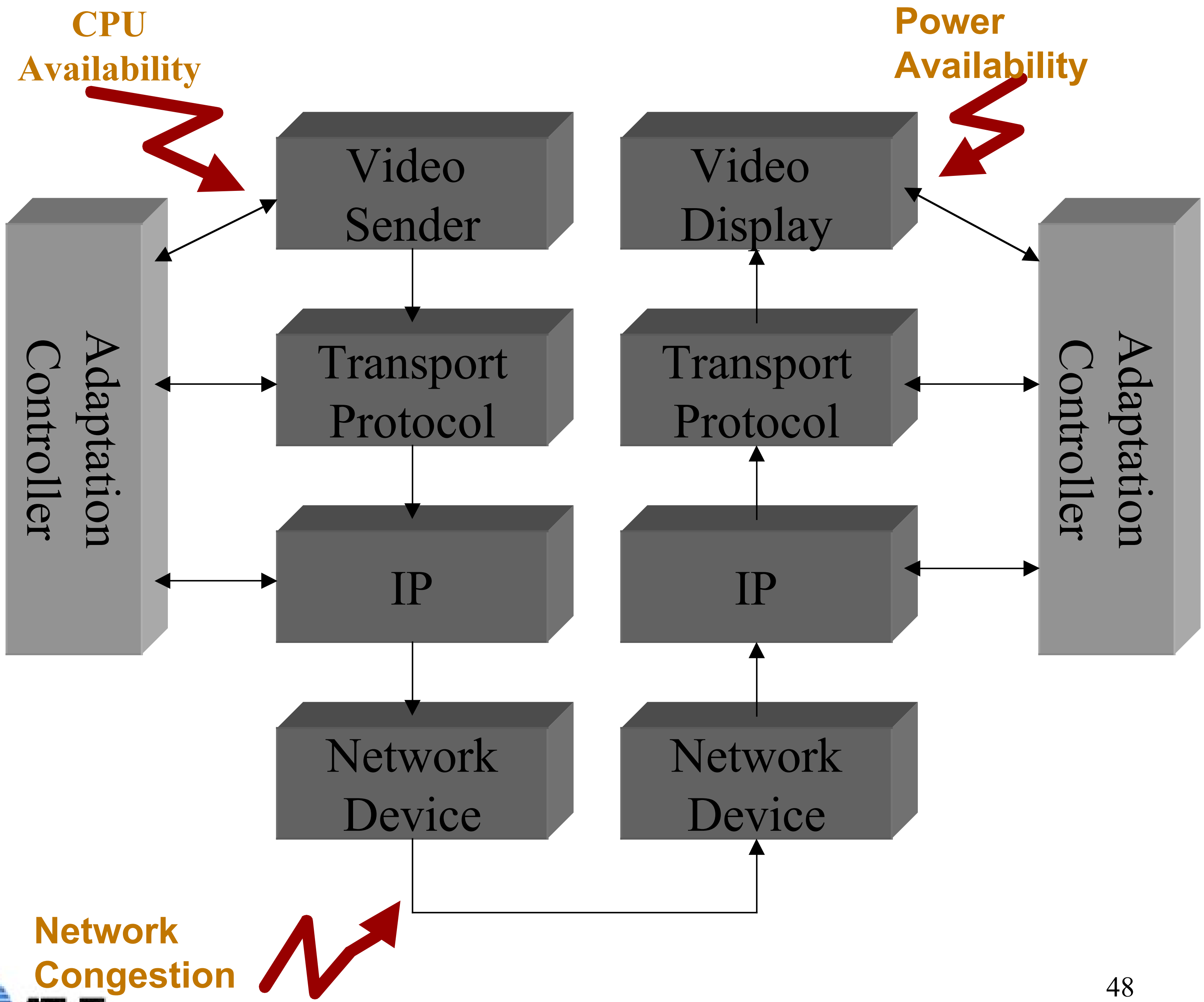
Ability to adapt is not enough:

- Adaptation policy
- Inter-component and inter-host coordination

Solution: Cholla coordination architecture

- Composable adaptation logic for composable software


CPU Availability
Power Availability
Video Sender
Video Display
Transport Protocol
Transport Protocol
IP
IP
Network Device
Network Device
Network Congestion


CPU Availability
Power Availability
Video Sender
Video Display
Transport Protocol
Transport Protocol
IP
IP
Network Device
Network Device
Adaptation Controller
Adaptation Controller
Network Congestion

Adaptation Controllers

Goal:

Compose and coordinate multiple adaptive components using composable controllers:

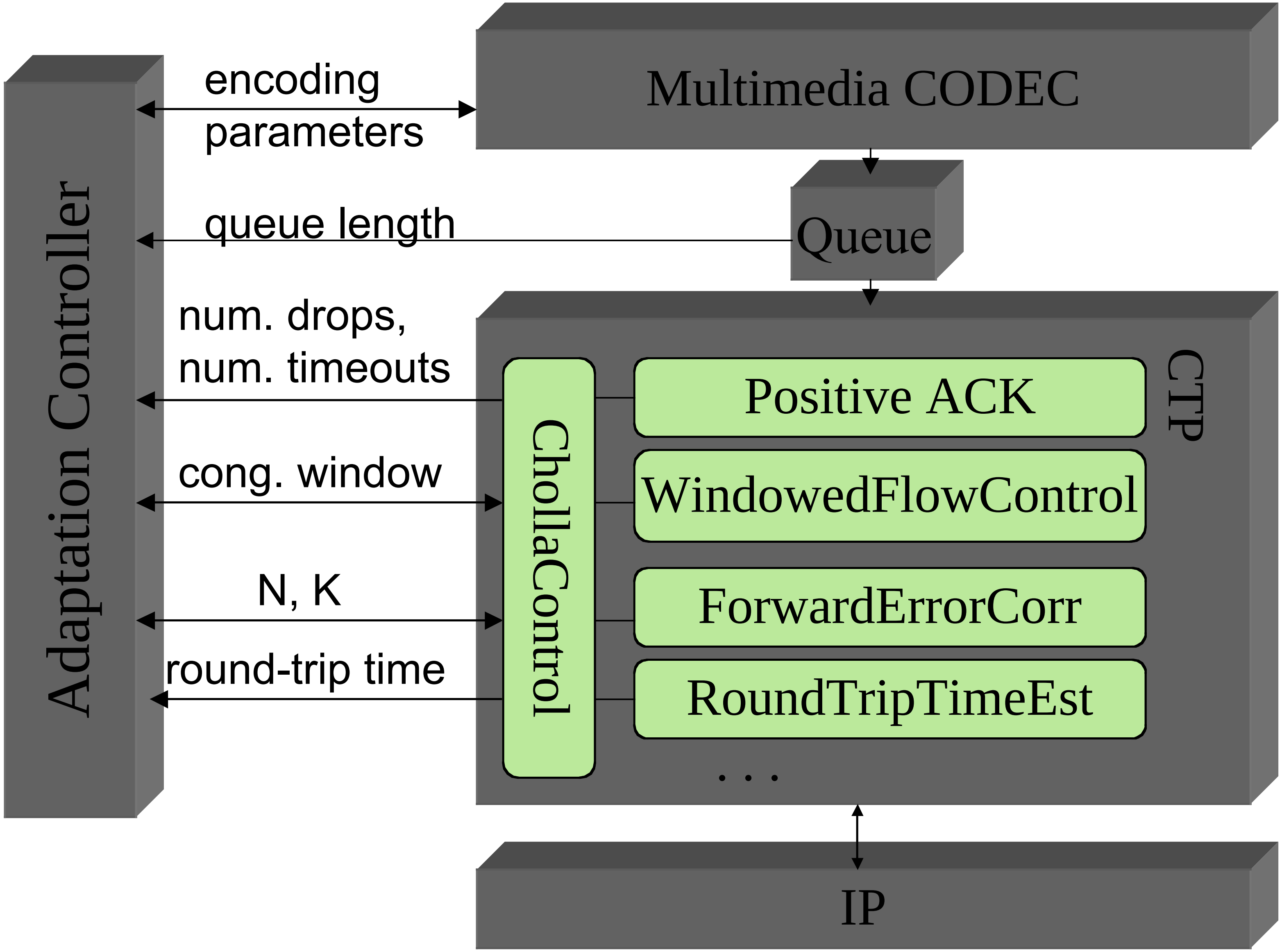
- Adaptive components controlled by adaptation policies
- Want to compose and coordinate fine-grained policies into a controller.
- Choose appropriate policies based upon:
  - User preferences (e.g. change framerate or picture quality)
  - Application demands (e.g. bandwidth or jitter sensitivity)
  - System requirements (e.g. wireless vs. wired network)

Expressing adaptation policies:

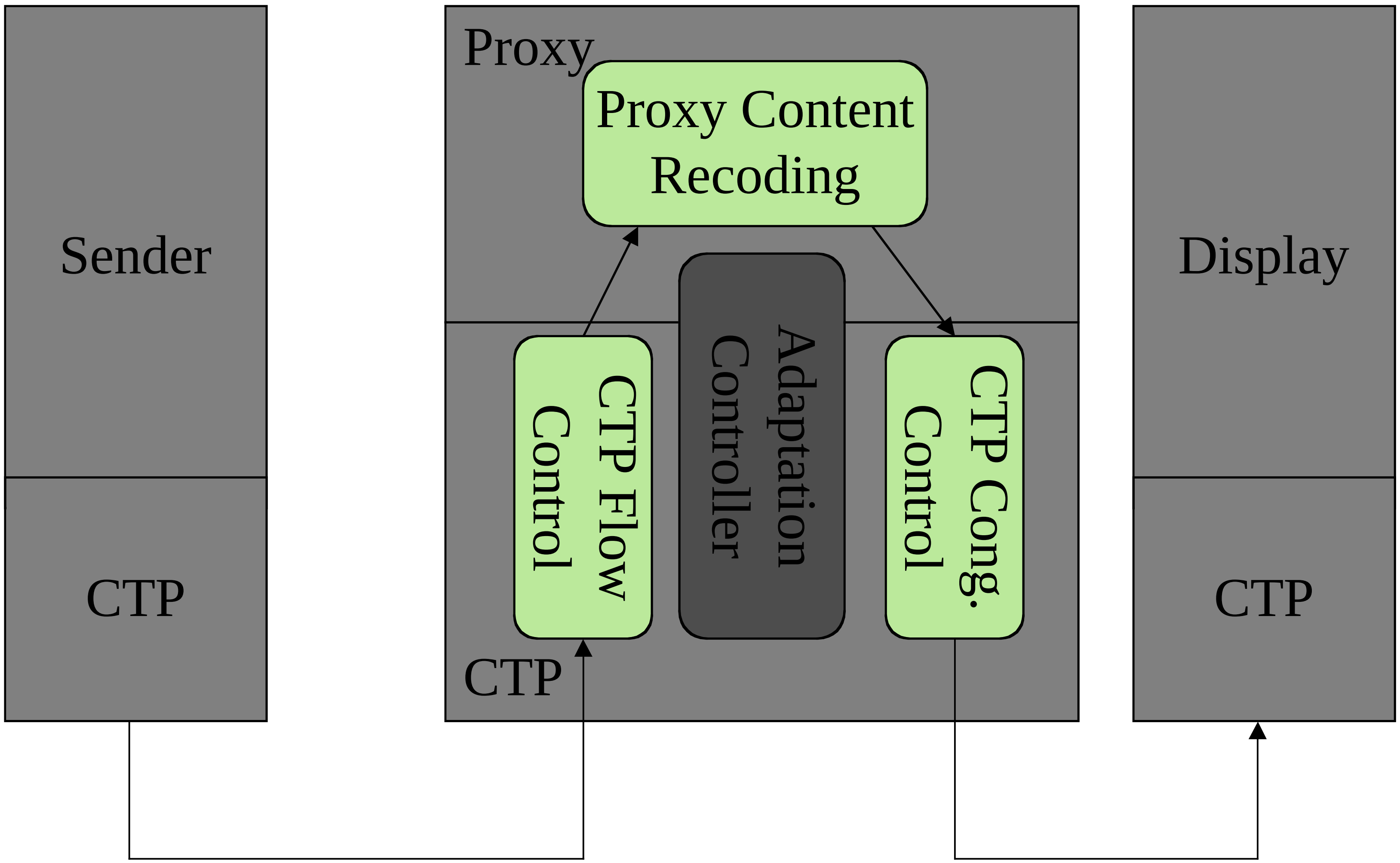
- Rule-based approach to constructing controllers
- A set of rules defines a particular behavior
- Sets of rules are composed into a controller that describe the policy

Coordination:

- Express explicit coordination policies as separate rulesets that govern the interactions between other policies
- Allow implicit coordination by exposing state of one component to other components


Multimedia CODEC
encoding parameters
Queue
queue length
Adaptation Controller
num. drops, num. timeouts
CTP
Positive ACK
WindowedFlowControl
ForwardErrorCorr
RoundTripTimeEst
CholläControl
cong. window
N, K
round-trip time
. . .
IP

Network Proxy Example



Inter-host coordination

Issues:

- Agreement on global state and need to adapt
- Synchronization of adaptation steps without violating the service properties

Work in progress:

- Libraries of reusable adaptation protocols that preserve different sets of service properties.
- GAP: Graceful Adaptation Protocol (ICDCS 01)

Conclusions

Dependability:

- Middleware must be designed carefully if it is to increase the application dependability

Adaptivity:

- Mechanisms are often relatively easy
- Policy and coordination issues often difficult
- Adaptation coordination architecture

For more information

iMobile: http://www.research.att.com/sw/tools/amn/

- Y.-F. Chen, H. Huang, R. Jana, T. Jim, M. Hiltunen, R. Muthumanickam, S. John, S. Jora, and B. Wei, "iMobile EE – An Enterprise Mobile Services Platform". To appear in ACM Journal on Wireless Networks, 2003.

Self*:

- K. Fetzer and K. Hogstedt, "Self*: A Data-Flow Oriented Component Framework for Pervasive Dependability", To appear in WORDS 2003, January 2003.

Cactus: http://www.cs.arizona.edu/cactus/

CQoS:

- J. He, M. Hiltunen, M. Rajagopalan, and R. Schlichting, "Providing QoS Customization in Distributed Object Systems", Middleware 2001, pages 351-372, November 2001.
- Extended version to appear in SPE, 2003.

Cholla:

- P. Bridges, "Composing and Coordinating Adaptations in Cholla", PhD Dissertation, University of Arizona, Department of Computer Science, Tucson, AZ, Dec 2002.