The Lost Art of Abstraction

Building Foundations for Dependable Distributed Systems

Rick Schlichting
Software Systems Research Department
AT&T Labs-Research

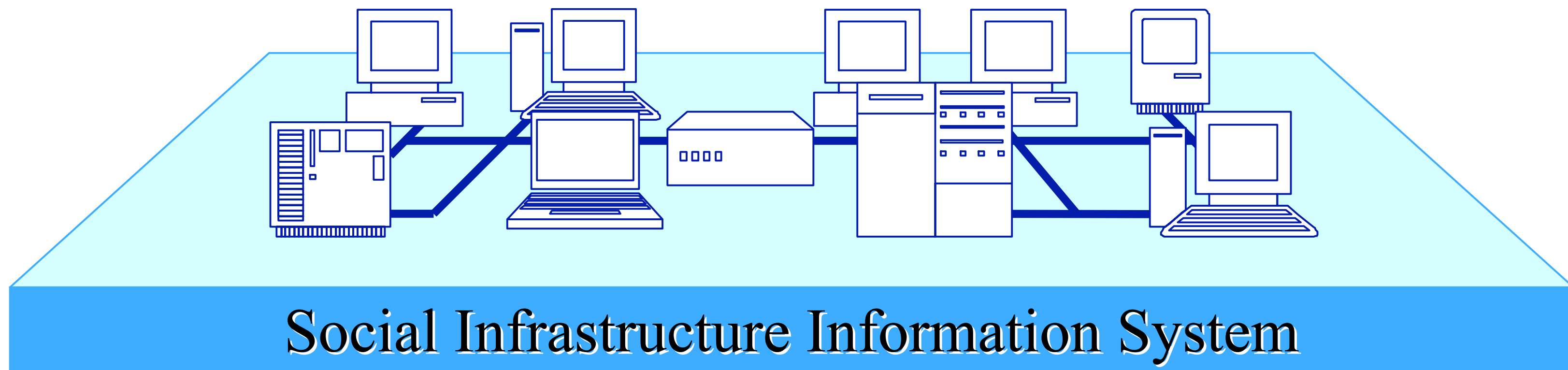
Work done in collaboration with:

- Matti Hiltunen (AT&T)
- Former Arizona PhD students Jun He (Cisco), Patrick Bridges (U. New Mexico), and Mohan Rajagopalan (Intel).
- Arizona faculty member Saumya Debray.
- AT&T researcher Trevor Jim.
- UIUC PhD student Kaustubh Joshi (AT&T VURI intern) and faculty member Bill Sanders.

 © Copyright 2006 AT&T. All Rights Reserved

Motivation

- Society is increasingly based on information systems and networks.



 © Copyright 2006 AT&T. All Rights Reserved

Next Generation Information Infrastructure

Characteristics

- Large number of networked machines.
- Spectrum of network types and technologies: wired, optical, wireless, ….
- Spectrum of distances: personal-area, local-area, metro-area, wide-area,….
- Spectrum of devices: from sensors to mobile units to high end machines and clusters.
- Spectrum of applications.
- Dynamic execution conditions and resource demands.
- Multiple administrative domains.

➡ MUST be dependable!

© Copyright 2006 AT&T. All Rights Reserved

Dependability

Definition: The trustworthiness of a computing system such that reliance can be justifiably placed on the service it delivers.
(Laprie, *et al*., *Dependability: Basic Concepts and Terminology*, Springer-Verlag, 1992)

Includes many properties and attributes

- Reliability
- Availability
- Safety
- Security
- Timeliness

Non-functional or Quality of Service (QoS) attributes

- Focus is not on *what* gets done, but rather *how well.*

Immensely challenging to build software with these attributes!

- Failures, intrusions….
- Concurrent and non-deterministic execution
- Heterogeneous systems and networks
- Resource constraints
- Multiple administrative domains
- Scale

Dealing with multiple attributes makes it even harder.

➡ Fundamental issue is complexity.

© Copyright 2006 AT&T. All Rights Reserved
at&t

System Abstractions

***System abstractions* can simplify the process.**

Definition

- Simplified model of a real-life hardware/software component or function.
- Extracts essential features while omitting unnecessary detail.

Goal: Building blocks for constructing more complex systems.

Have long been used to as a way to simplify the design of complex systems.

"Classic" examples

- Process, file, virtual memory,….
- Layered operating system architectures (e.g., THE system).

➡ **Good abstractions are those that people use without thinking about the underlying implementation.**

© Copyright 2006 AT&T. All Rights Reserved

What about Dependability?

Certainly some good dependability-related abstractions

- Provide enhanced QoS characteristics.

Hardware virtualization

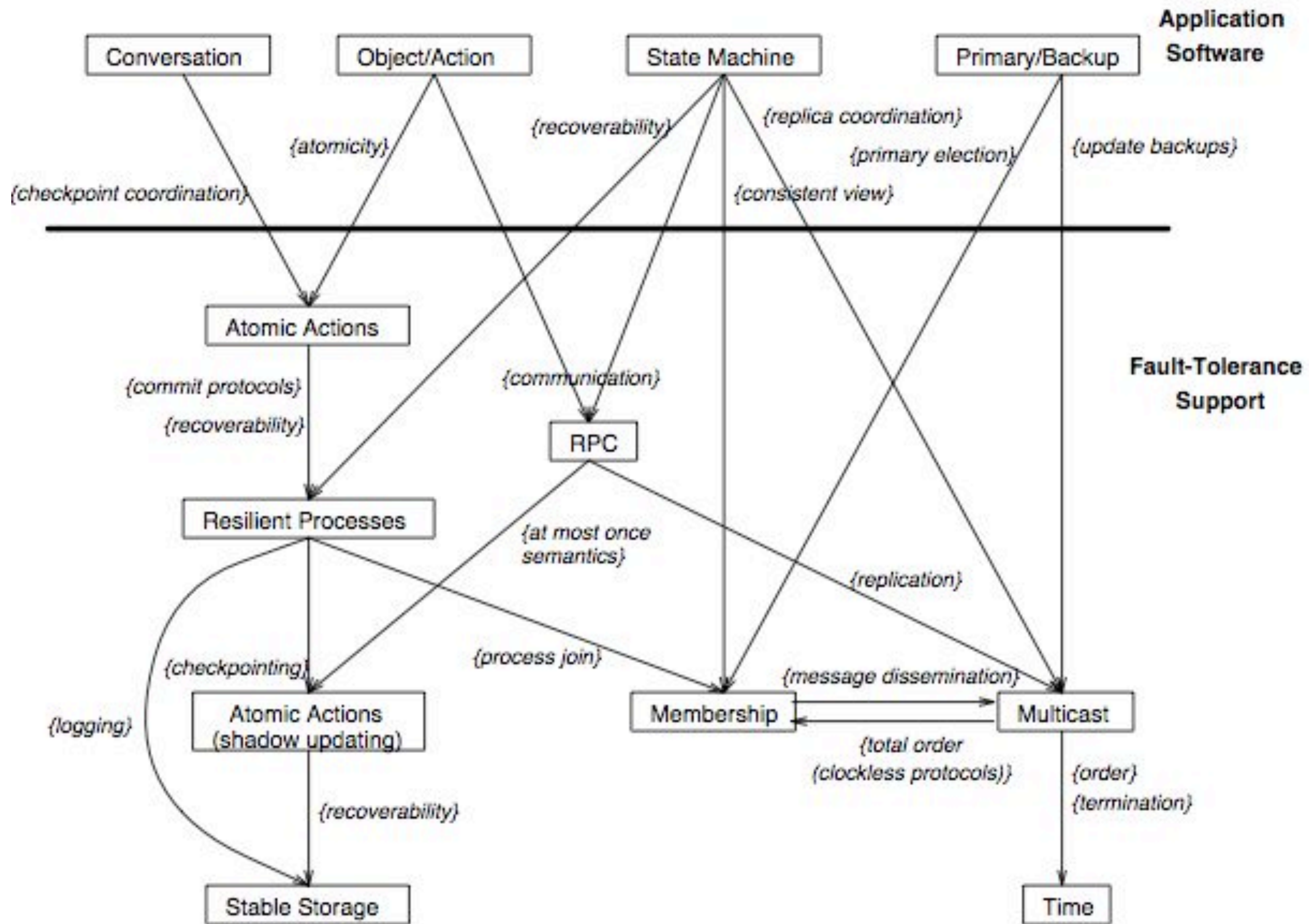
- Stable storage: abstract storage that never fails.
- Fail-stop processor: virtual processor whose only failure is a detectable crash.

Services for networked systems

- Often focus on providing common global information across machines despite machine and network failures (*virtual shared state*).
- Implemented as middleware and/or using network protocols.
- Consistent global clock: abstraction of a single system-wide clock.
- Atomic multicast: shared message queue.
- Distributed atomic actions (transactions): all or nothing execution across machines.

Can also be organized as layers or hierarchies.

© Copyright 2006 AT&T. All Rights Reserved
at&t



© Copyright 2006 AT&T. All Rights Reserved
at&t

Challenges and Issues

Abstraction failures (*leaky abstractions*)

- Impossible to implement an abstraction in which QoS properties hold under all conditions.
- Inherently probabilistic.

Composing abstractions

- Reasoning about properties of combinations of abstractions.
- Conflicts and tradeoffs between different attributes.
- Performance overhead.

Unnecessary attributes

- Matching attributes of abstractions to application and execution environment.
- Unnecessary attributes can mean extra execution overhead.

Mechanism-oriented design

- Focus on mechanism rather than abstraction.
- Protocols (e.g., SOAP), survivable systems (e.g., IDSs).

Changing QoS attributes dynamically

- Providing ability to adapt at runtime

 © Copyright 2006 AT&T. All Rights Reserved

Ideas

Translucent abstractions

- Explicitly exposes useful information about internal operation.
- Would be useful, for e.g., for TCP operation over wireless links.
- Example: *accrual failure detectors*, which gives an estimate of the probability that a host has failed rather than just a binary indication.

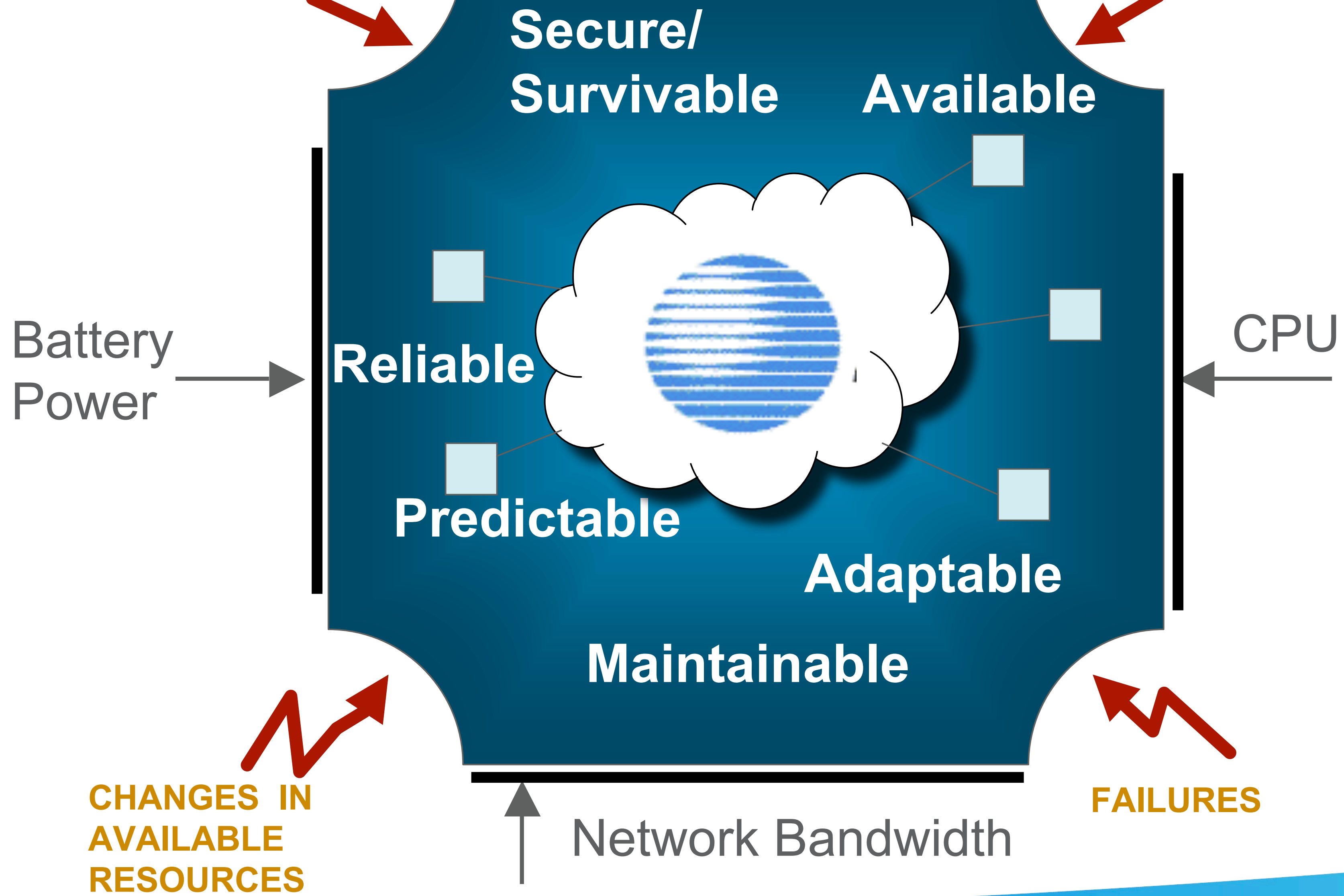
Customizable and synthesized abstractions

- Allows the attributes and levels of assurance to be customized based on application requirements and execution environment.

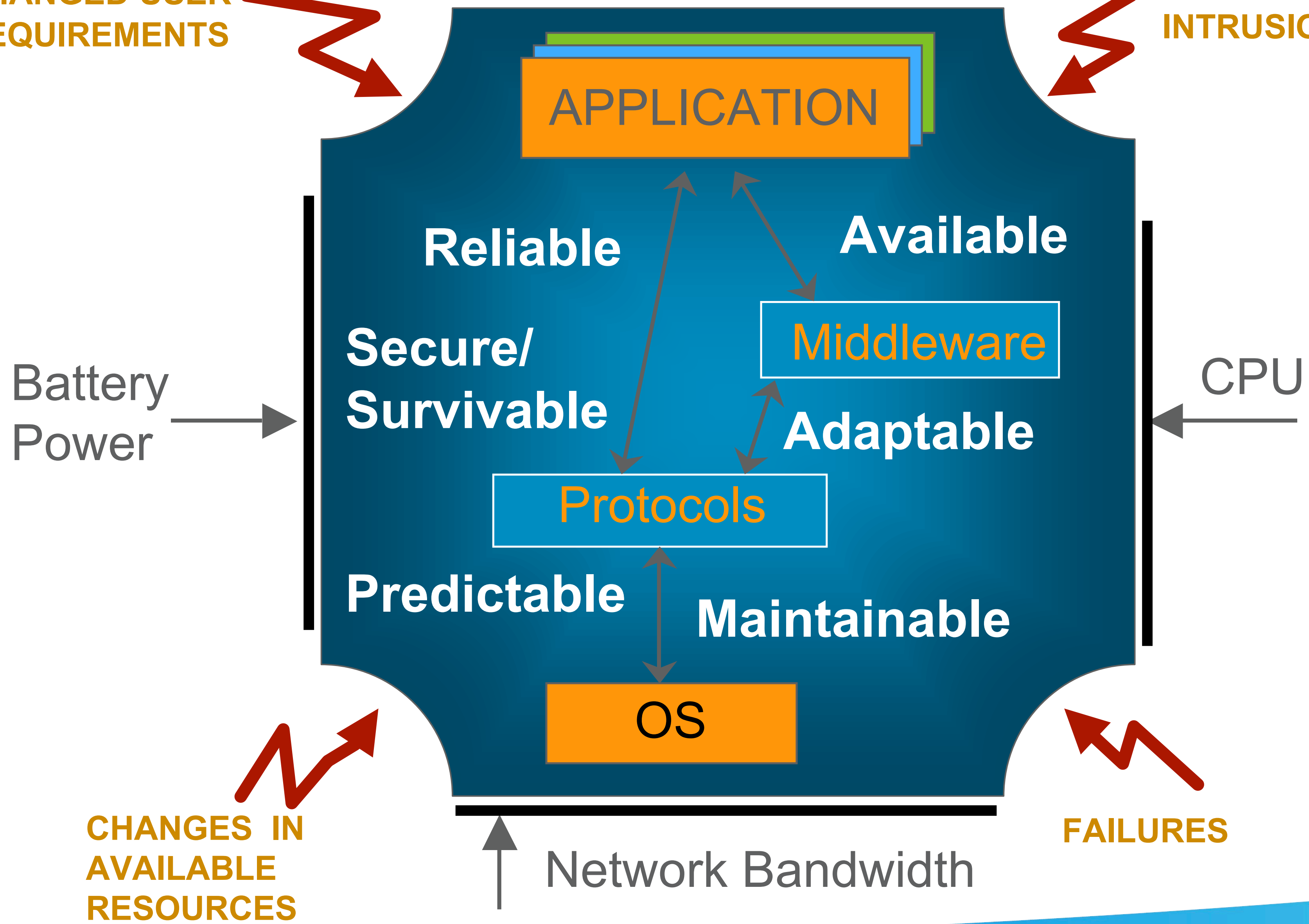
Abstractions for survivability

- *Instrusion-stop process*, which stops executing and issues a notification when compromised.

 © Copyright 2006 AT&T. All Rights Reserved



 © Copyright 2006 AT&T. All Rights Reserved



 © Copyright 2006 AT&T. All Rights Reserved

Dependable Systems Research at AT&T

Provide support for building system abstractions and services that bridge the gap between network and application.

Support for configurable solutions

- Ability to customize properties to the characteristics of the execution environment and the needs of the application.

Support for adaptive behavior

- Ability to change execution behavior dynamically to react to changes in the execution environment or the application.

Support for synthesized solutions

- Ability to synthesize abstractions that optimize system attributes such as performance or dependability (*holistic optimization*).

Cactus ⇨ configuration

Cholla ⇨ adaptation

Cassyopia ⇨ synthesis

 © Copyright 2006 AT&T. All Rights Reserved

Cactus: Building Highly Configurable Software

Both a programming model and an implementation framework for building customized software from collections of software modules.

Highlights

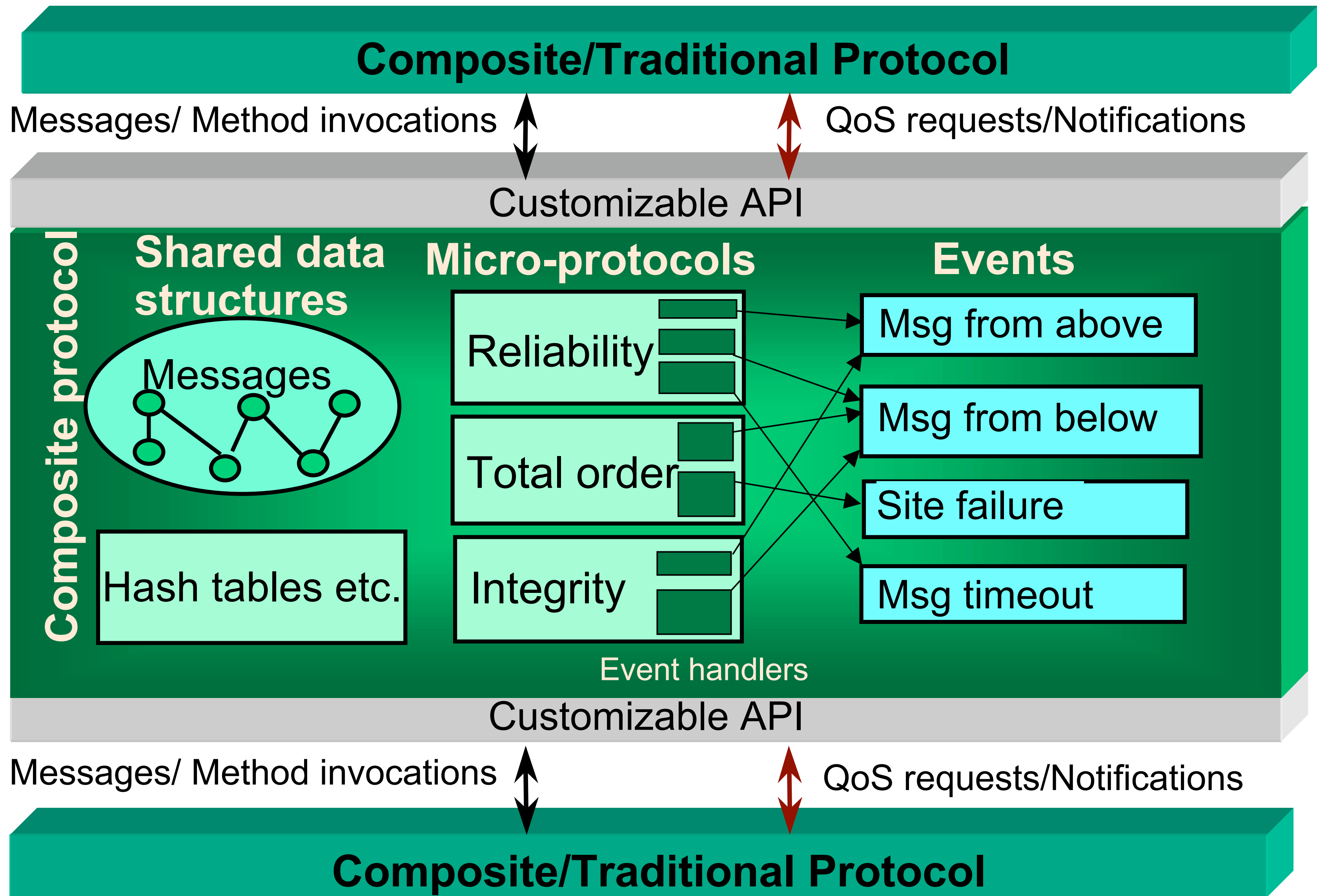
- Fine-grain configuration and customization.
- Multiple types of attributes and properties, each implemented by a collection of alternative modules.
- Combination of hierarchical and non-hierarchical composition.

Focus

- Communication-oriented services in networks, i.e., protocol stacks and distributed services (but more general).
- Highly customizable Quality of Service (QoS) attributes related to fault tolerance, timeliness, security, etc. (but useful for other reasons).

Addresses challenge of module interaction in highly-configurable software.

© Copyright 2006 AT&T. All Rights Reserved



 © Copyright 2006 AT&T. All Rights Reserved

Cactus Model

Protocol/service = composite protocol

- Provides service-specific API.

Property/QoS attribute = micro-protocol (MP)

- MPs interact using an events, shared data, and *dynamic messages*.
- Mechanisms provide decoupling of MPs ⇒ configurability.

Service customization = choose appropriate MPs

Dynamic adaptation = load/activate/deactivate MPs at runtime

Two implementations of Cactus 3.0

- C version running on different variants of Unix.
- Java version.

 © Copyright 2006 AT&T. All Rights Reserved

Example Protocols and Services

Configurable Transport Protocol (CTP)

- Ordering, reliability, flow/congestion control, security.

Secure and Survivable Communication (SecComm)

- Privacy, authenticity, integrity, replay prevention, combinations.

Configurable Quality of Service (CQoS)

- Adding transparent multi-dimensional QoS customization to distributed object systems.

Distributed System Monitoring Service (CDSMon)

- Function to be monitored.

Location-Based Services (LBS)

- Functionality based on location for mobile services.

Ad-Hoc Networking (AHN)

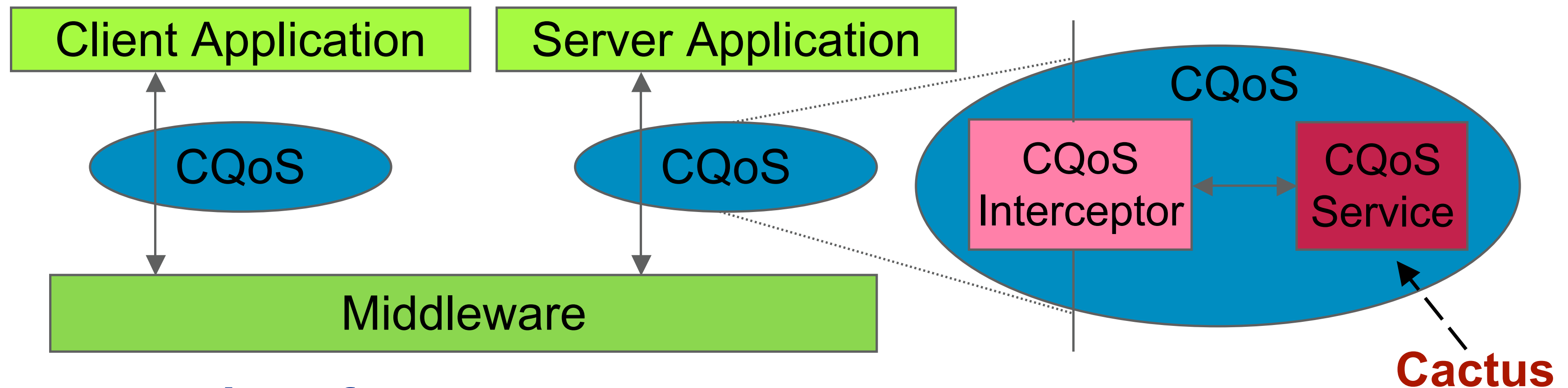
- Dynamic QoS

AT&T Enterprise Messaging Network (EMN)

- Per request QoS for mobile service platforms

 © Copyright 2006 AT&T. All Rights Reserved

CQoS Architecture (J. He)



CQoS consists of two components

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

Micro-protocols include

Fault tolerance: ActiveRep, PassiveRep, TotalOrder, MajorityVote, Membership, StateRecovery… .

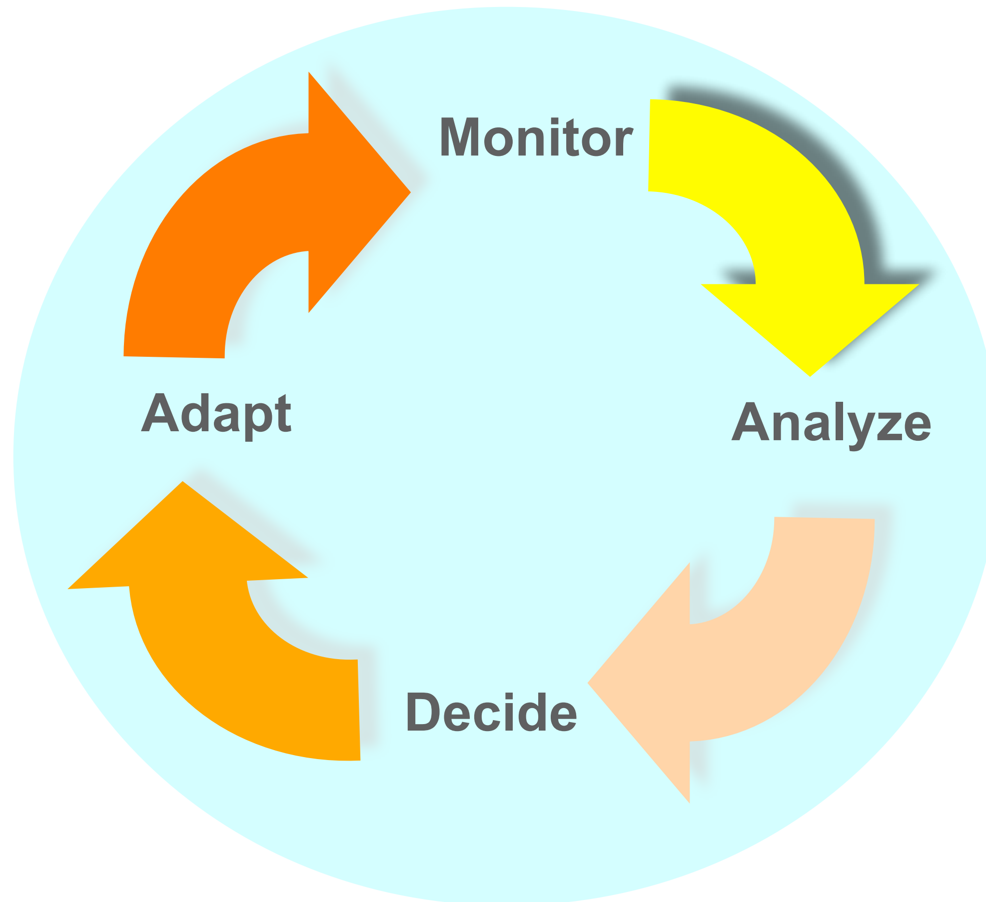
Security: DESPrivacy, Authentication, AccessControl …

Timeliness: PrioritySched, QueueSched, TimedSched.

Semantically different combinations of micro-protocols provide semantically different variations of multi-dimensional QoS.

© Copyright 2006 AT&T. All Rights Reserved
at&t

Adaptive Systems



Each phase can be complex in large networked systems

- Monitoring involves data across multiple hosts and multiple sources.
- Analyzing may involve heuristics or evaluation over time.
- Decision may involve evaluating tradeoffs or distributed algorithms.
- Adaptation may involve distributed coordination across multiple hosts.

All must be done in a running system and an environment that continues to change.

Adaptation mechanisms versus policies

- Mechanisms provide hooks for monitoring and effecting changes as well as protocols for data collection, analysis, and adaptation coordination.
- Policy encapsulates tradeoff analysis and "business logic".

 © Copyright 2006 AT&T. All Rights Reserved

Cholla Adaptation Architecture (P. Bridges)

Challenges

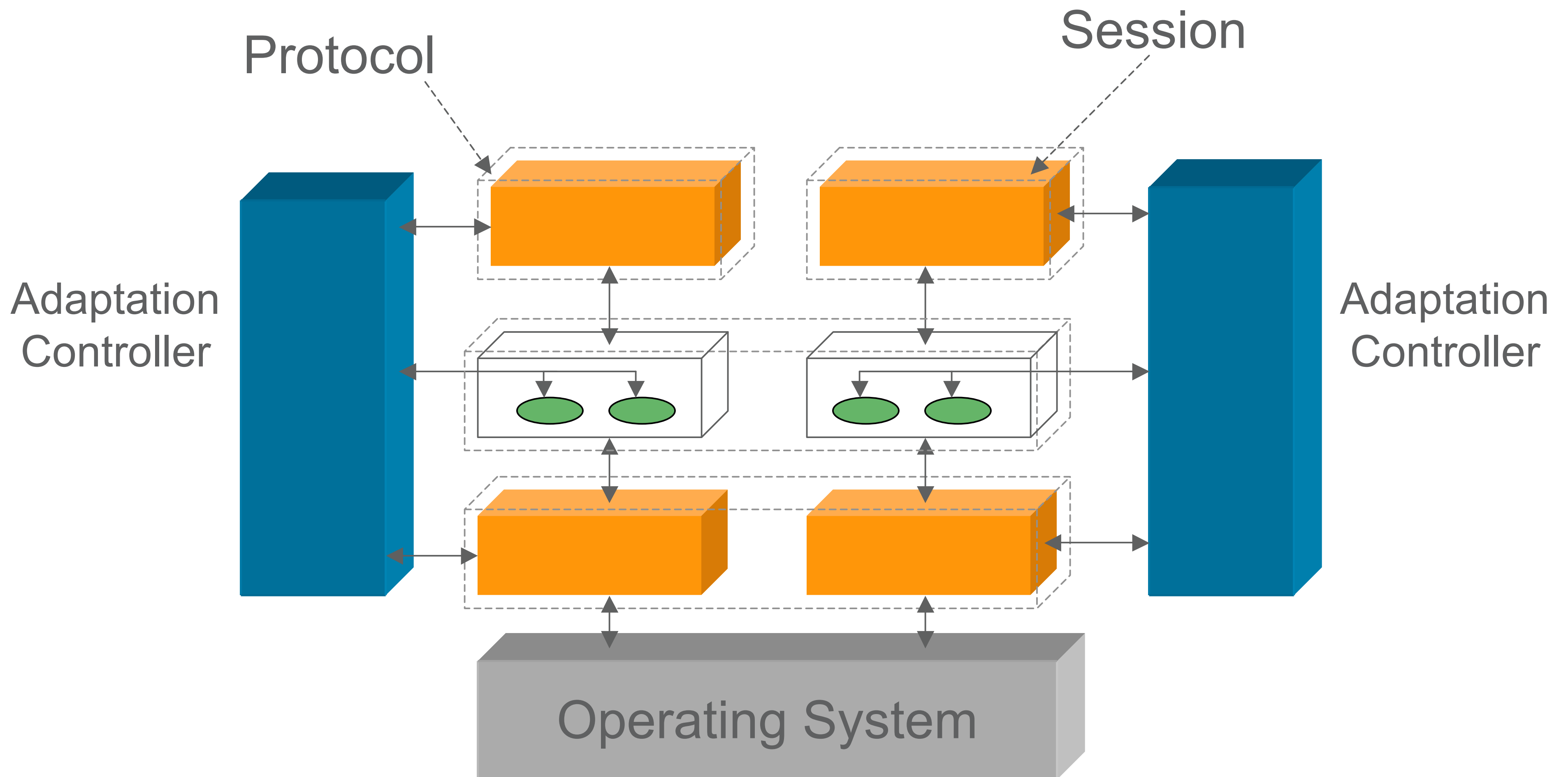
- Decoupling control from regular functionality.
- Coordinating adaptations
  - Inter-component coordination on a single host
  - Inter-host coordination for distributed services
- Composition of adaptation policies.
- Developing appropriate adaptation policies.
- Efficient realization of policies.

Solution: Cholla adaptation architecture

- Uses Cactus as underlying platform for implementing adaptive mechanisms and protocols.

© Copyright 2006 AT&T. All Rights Reserved

Software Architecture



© Copyright 2006 AT&T. All Rights Reserved

Adaptation Controller

Implements execution feedback control loop:

- Monitors system state and controls adaptation.

Monitoring:

- Input variables from controlled components.
- Input from external monitoring.

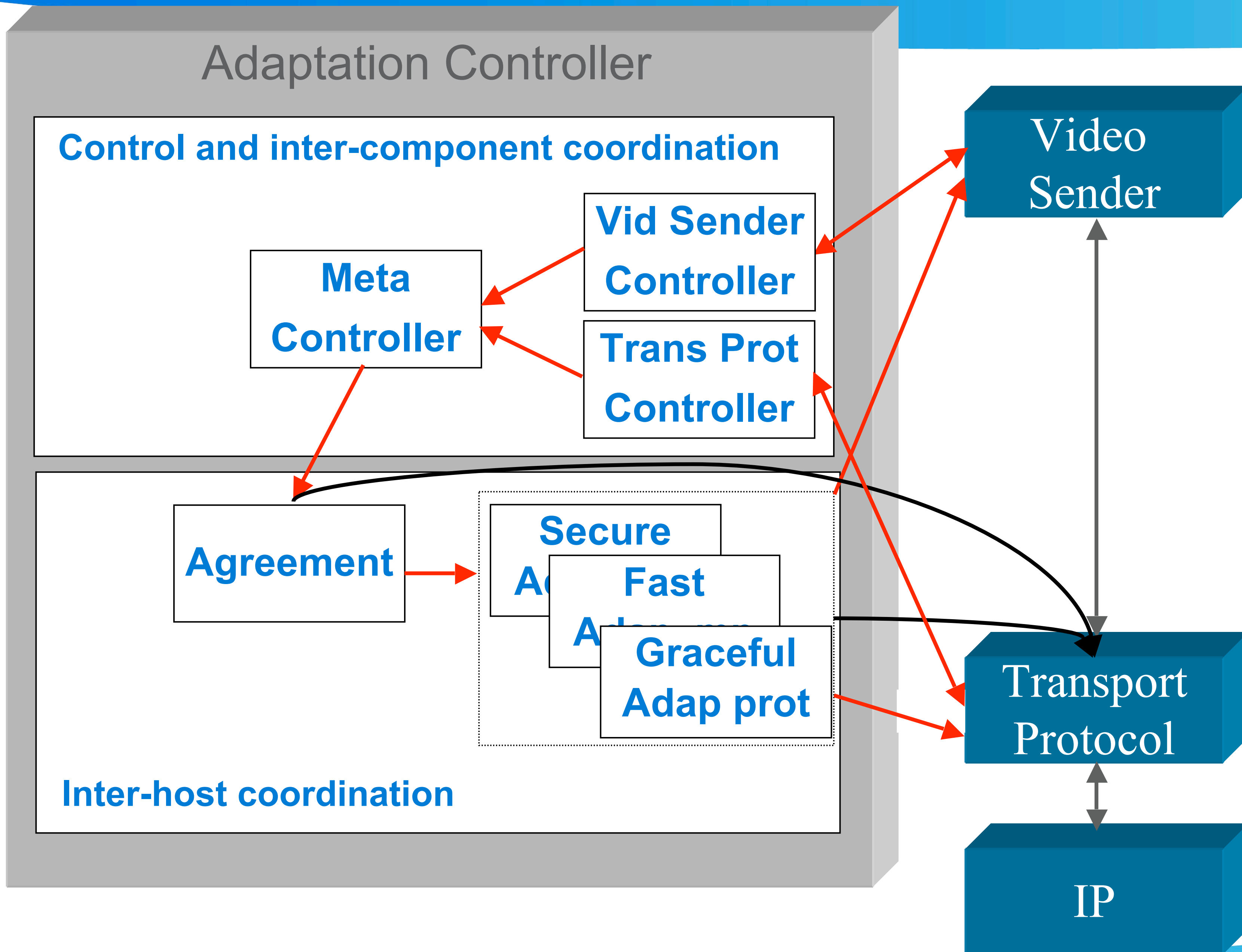
Control:

- Generates outputs based on inputs plus adaptation policies.
- Changes execution parameters in controlled components (value adaptations).
- Orchestrates module changeovers (algorithmic adaptations).

Implementations:

- FLAC: Fuzzy logic based adaptation controller. Focuses on value adaptations and inter-component coordination.
- CAC: Cactus based adaptation controller. Focuses on algorithmic adaptations and inter-host coordination.
- Others possible….

 © Copyright 2006 AT&T. All Rights Reserved

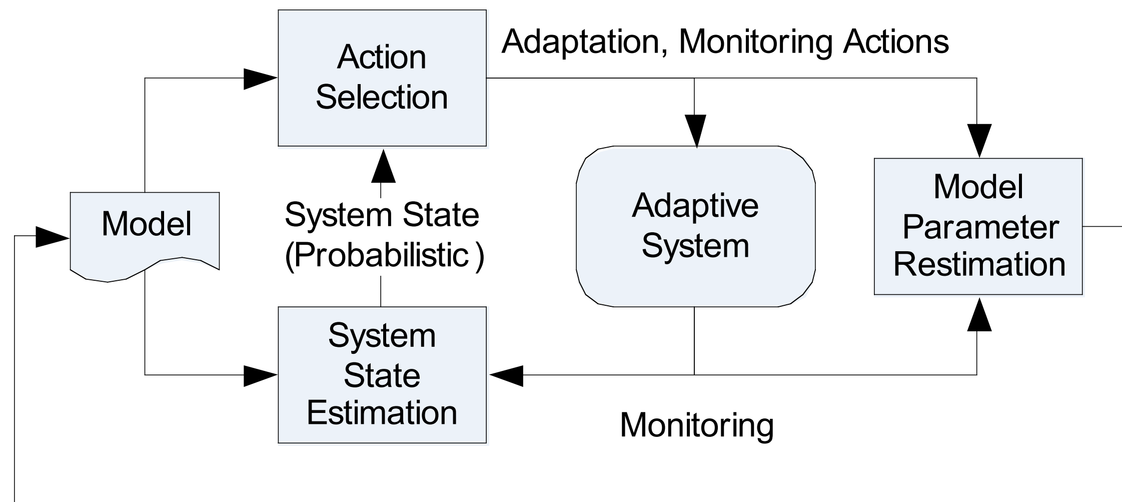


 The Lost Art of Abstraction © Copyright 2006 AT&T. All Rights Reserved

at&t

Policy Generation (K. Joshi, W. Sanders)

Goal: Use stochastic models of system and environment to generate good policies for selecting adaptive actions.



Use Bayesian Techniques for State-Estimation

Multiple Algorithms for Action Selection (control)

- Single-step (greedy): look at the effects of next action only to determine adaptation
- Multi-step: treat problem as a sequential decision problem; choose adaptations by looking for best sequences of adaptation actions

Applied to Automatic Distributed System Recovery

© Copyright 2006 AT&T. All Rights Reserved
at&t

Cassyopia: Synthesizing Abstractions

(M. Rajagopalan, S. Debray)

***Holistic system optimization*: consider the system as an integrated whole.**

Goals:

- Increase the scope of optimization, e.g., across address spaces.
- A uniform approach that generalizes across metrics, e.g., performance and dependability.
- Based on compiler optimization techniques and binary rewriting infrastructure.

Examples:

- Event-based systems [PLDI 2002]
- System call clustering [In submission]
- Authenticated system calls [DSN DCCS 2005]

➡All can be viewed as synthesizing new abstractions automatically using compiler techniques.

© Copyright 2006 AT&T. All Rights Reserved

System Call Clustering

System calls are ubiquitous but still expensive.

Profiling to identify system calls that can be executed in a single kernel crossing ➡ *system call cluster*:

- Non linear sequences
- Across function boundaries

Maximize size of cluster through compiler techniques:

- Code motion
- Function inlining
- Loop unrolling

multi-call : new OS primitive that allows multiple system calls in a single boundary crossing.

Experimental results:

- mpeg_play 20% frame rate, 15% execution time

© Copyright 2006 AT&T. All Rights Reserved

Authenticated System Calls
(M. Rajagopalan, T.Jim)

New implementation of a *system call monitor*.

Observation: Attacker often use system calls to inflict real damage a a system.

Authenticated system call

- New OS primitive that can monitor and enforce system call policies
- Regular system calls with additional parameters

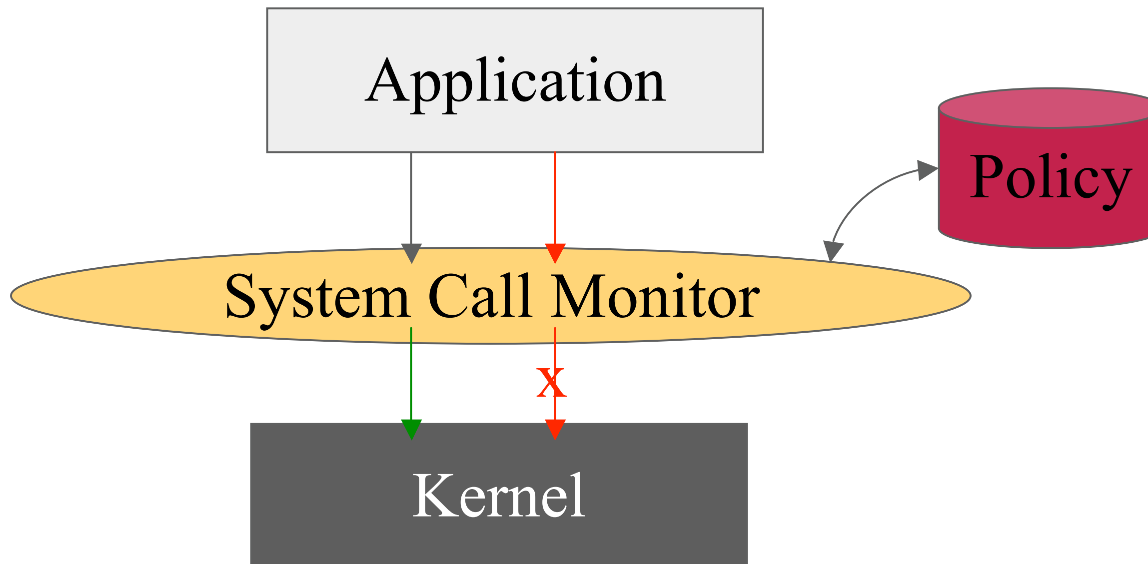
  Policy : specifies expected system call behavior

  MAC : cryptographically guarantees integrity of system call and arguments
- Executed only if the call conforms with the specified policy

Compiler techniques to generate policies and to transform binaries to synthesize new calls.

 © Copyright 2006 AT&T. All Rights Reserved

System Call Monitoring



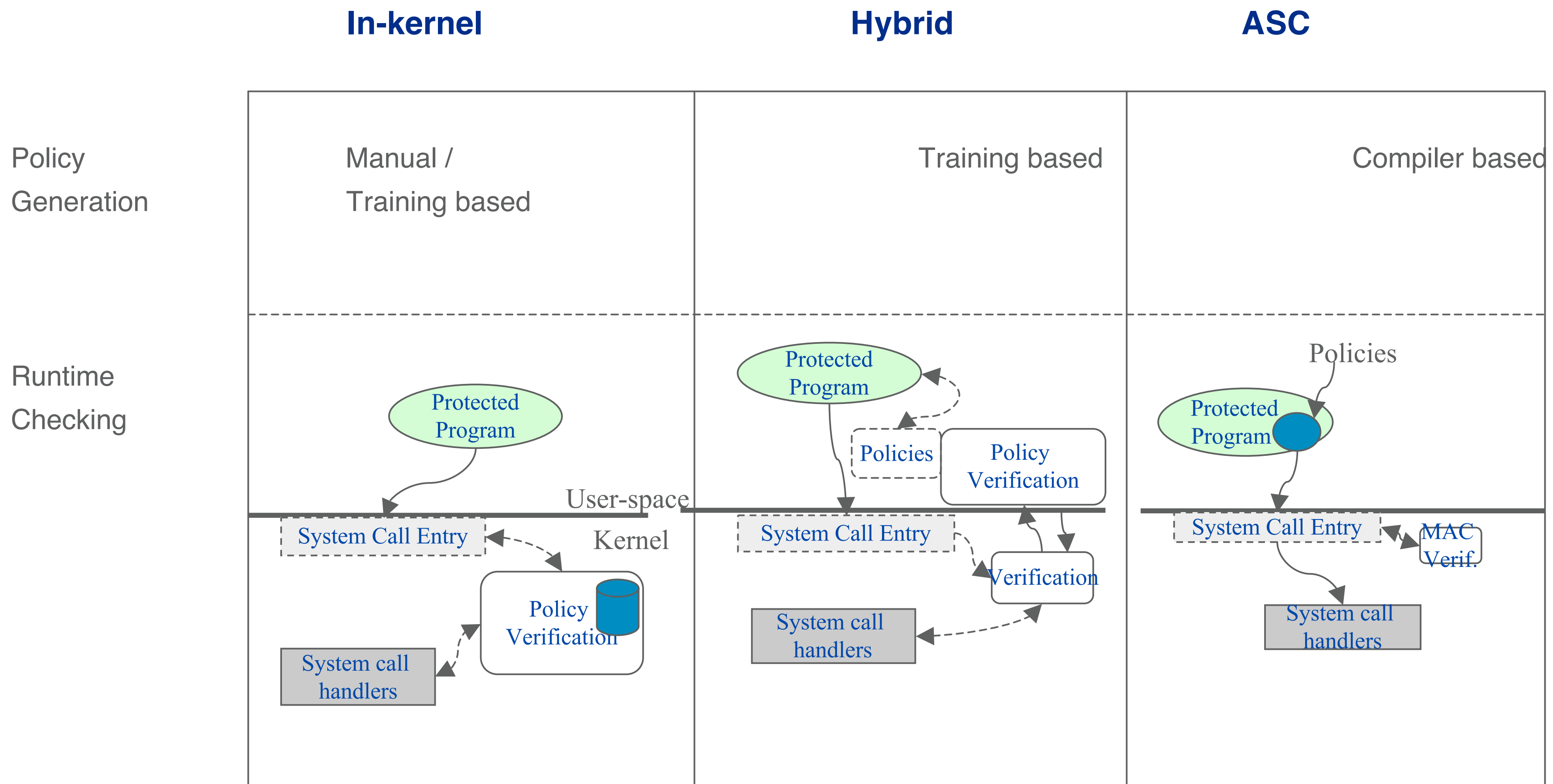
POLICY

```
native-__sysctl: permit
native-break: permit
native-chdir: permit
native-close: permit
native-execve: filename eq "/usr/bin/sudo" then deny
native-execve: filename match "/bin/*" then permit
```

Snippet from systrace policy for /bin/ksh (www.hairyeyeball.org)

 © Copyright 2006 AT&T. All Rights Reserved

Comparing Implementation Strategies



Mapping policy to program
Increased kernel complexity

Mapping policy to program
Performance
Race conditions (TOCTOU Attacks)

 © Copyright 2006 AT&T. All Rights Reserved

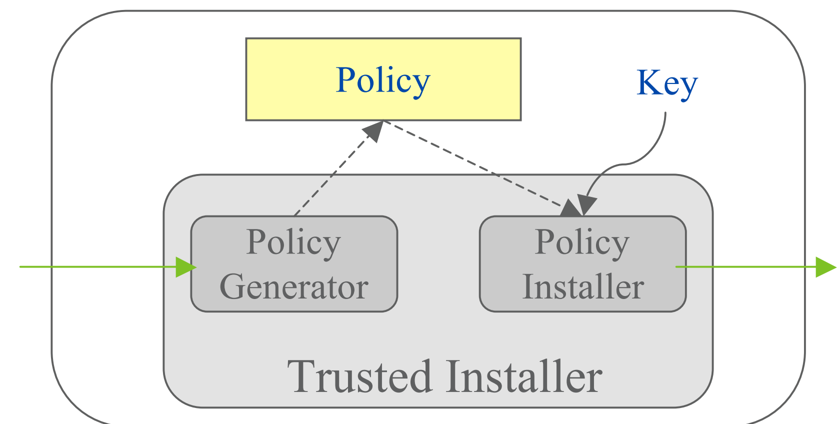
at&t

Generating Policies

Trusted Installer based on PLTO binary rewriting system.

Steps

- Disassembly, IR
- Policy generation
- Create ASC
- Replace syscall with ASC
- Reassemble, rewrite



Advantages

- Completely automatic, < 30sec for programs in Spec2000 suite
- Does not miss rarely used calls

The Lost Art of Abstraction © Copyright 2006 AT&T. All Rights Reserved

at&t

System Call Policies

Policy: **Set of verifiable properties of the system call request.**

Basic policy **contains**

- System call number
- System call site
- Some argument values

For example

open("/dev/console",0x5)

Permit *open* from location 0x806c462

Parameter 0 equals "/dev/console"

Parameter 1 equals 5

Control flow policy

- Constrain sequence of system calls in a program.

 © Copyright 2006 AT&T. All Rights Reserved

at&t

ASCs and Policy Enforcement

An ASC is a regular system calls with additional parameters

Policy : bit string encoding expected system call behavior

MAC : cryptographically guarantees integrity of system call and arguments

open("/dev/console",0x5) → open("/dev/console",0x5,policy,MAC)

Policy enforcement

- When a system call occurs:
  - Create new encoded policy (EP') based on policy argument.
- Compute the MAC' of EP'.
- Allow call only if MAC' is the same as MAC passed as argument.
- Any tampering with the system call will cause MACs to differ.

 © Copyright 2006 AT&T. All Rights Reserved

at&t

Conclusions and Future Work

Useful system abstractions are the key to building a highly dependable information infrastructure.

Our research is addressing issues related to building such abstractions

- Cactus: Flexible configuration based on two-level composition model.
- Cholla: Control and coordinated adaptation.
- Cassyopia: Compiler techniques for synthesizing new mechanisms.

Future work

- Using Cactus and protocols/services built using Cactus.
- Continue synthesis work.
- Applications, applications, applications!
- Policies, policies, policies!

 © Copyright 2006 AT&T. All Rights Reserved

For More Information

Bhatti, Hiltunen, Schlichting, and Chiu. Coyote: A System for Constructing Fine-Grain Configurable Communication Services. *ACM Trans. on Computer Systems* 16, 4 (Nov. 1998), 321-366.

Chen, Hiltunen, and Schlichting, Constructing Adaptive Software in Distributed Systems. *Proc. 21st Conf. on Distributed Computing Systems,* (April 2001), 635-643.

Hiltunen, Schlichting, and Ugarte. Building Survivable Services using Redundancy and Adaptation, *IEEE Trans. on Computers* (February 2003), 181-194.

He, Hiltunen, Rajagopalan, and Schlichting. Providing QoS Customization in Distributed Object Systems, *Software-Practice and Experience* 33,4 (April 2003), 295-320.

Joshi, Hiltunen, Schlichting, Sanders, and Agbaria. Online Model-Based Adaptation for Optimizing Performance and Dependability. *Proc. ACM SIGSOFT Workshop on Self-Managed Systems* (Oct. 2004).

Rajagopalan, Hiltunen, Jim, Schlichting. Authenticated System Calls. *Proc. DSN-2005 Dependable Computing and Communication Symp.* (June 2005), 358-367.

Hiltunen, Schlichting. The Lost Art of Abstraction. In *Architecting Dependable Systems III*, (R. de Lemos, C. Gacek, A. Romanovsky, ed.), Lecture Notes in Computer Science, Volume 3549, Springer-Verlag, Berlin, 2005, pp. 331-342.

Joshi, Hiltunen, Sanders, and Schlichting. Automatic Model-Driven Recovery in Distributed Systems. *Proc. 24th Symp. On Reliable Distributed Systems* (Oct. 2005), 25-38.

 © Copyright 2006 AT&T. All Rights Reserved

Thank you!