Open Group, RTES, Wed 30 Jan 2008

Separation and Integration in MILS
Or What is MILS? (and HAP?)
Toward MILS Definitions

John Rushby,

Computer Science Laboratory
SRI International
Menlo Park CA USA

Motivation

- The July 2007 “Layered Assurance” workshop
- Email requests
- Needed for “The Constitution”
- And for MILS Integration (MIPP)

Existing Ideas

- Bottom-up descriptions via partitioning, separation, SKPP
- Catalogs of components and examples
- Do without: "Component Security Integration"
- Top-down description: separation and integration

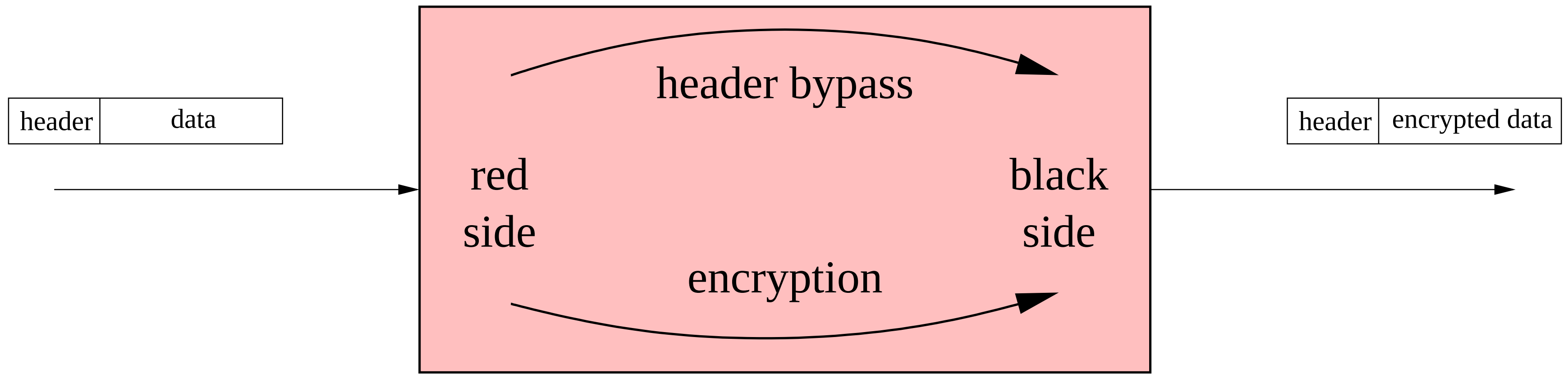
Top-Down Description

- MILS is a two-level approach
- Cf. classical policy-mechanism distinction
- **Policy Level:** perform a decomposition to a virtual architecture (circles and arrows picture) and identify the trusted components and their local policies
    - Do this in a way that minimizes complexity of trusted components and their policies
    - assume components and arrows are free

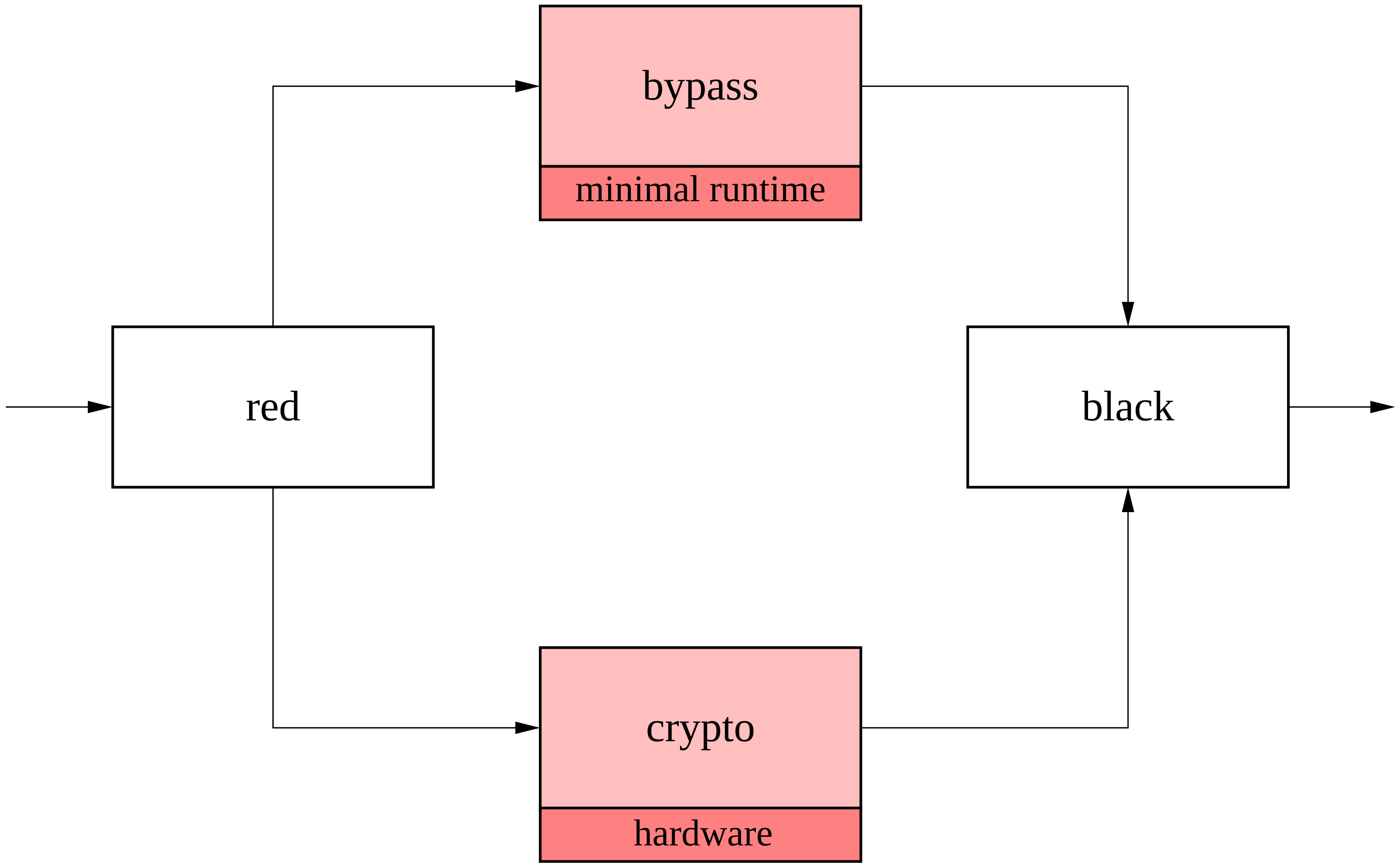
Resource Sharing Level: figure out how to allocate virtual components to physical resources

- MILS provides technologies (basically, separation) so that virtual components of various types can share physical resources without compromising the integrity of the policy level

Example: The Classical Crypto Controller



Policy-Level Decomposition

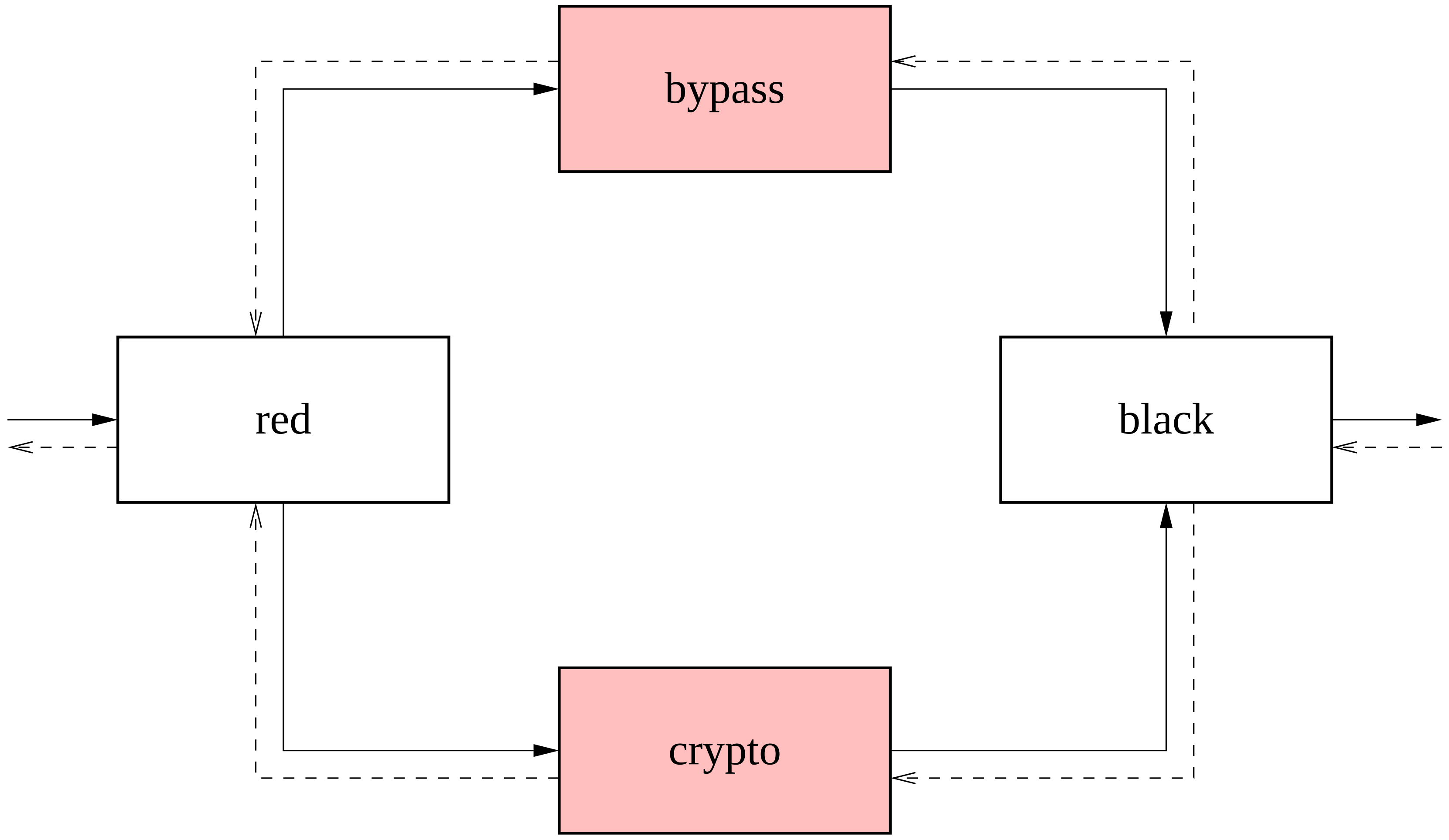


Underlying Model

- Arrows indicate interfaces
- Only interaction among components is through these explicitly shown interfaces
- Absence of arrows is often crucial
- Formally, circles/components are state machines, arrows are shared variables or queues (depending on the exact formal model)

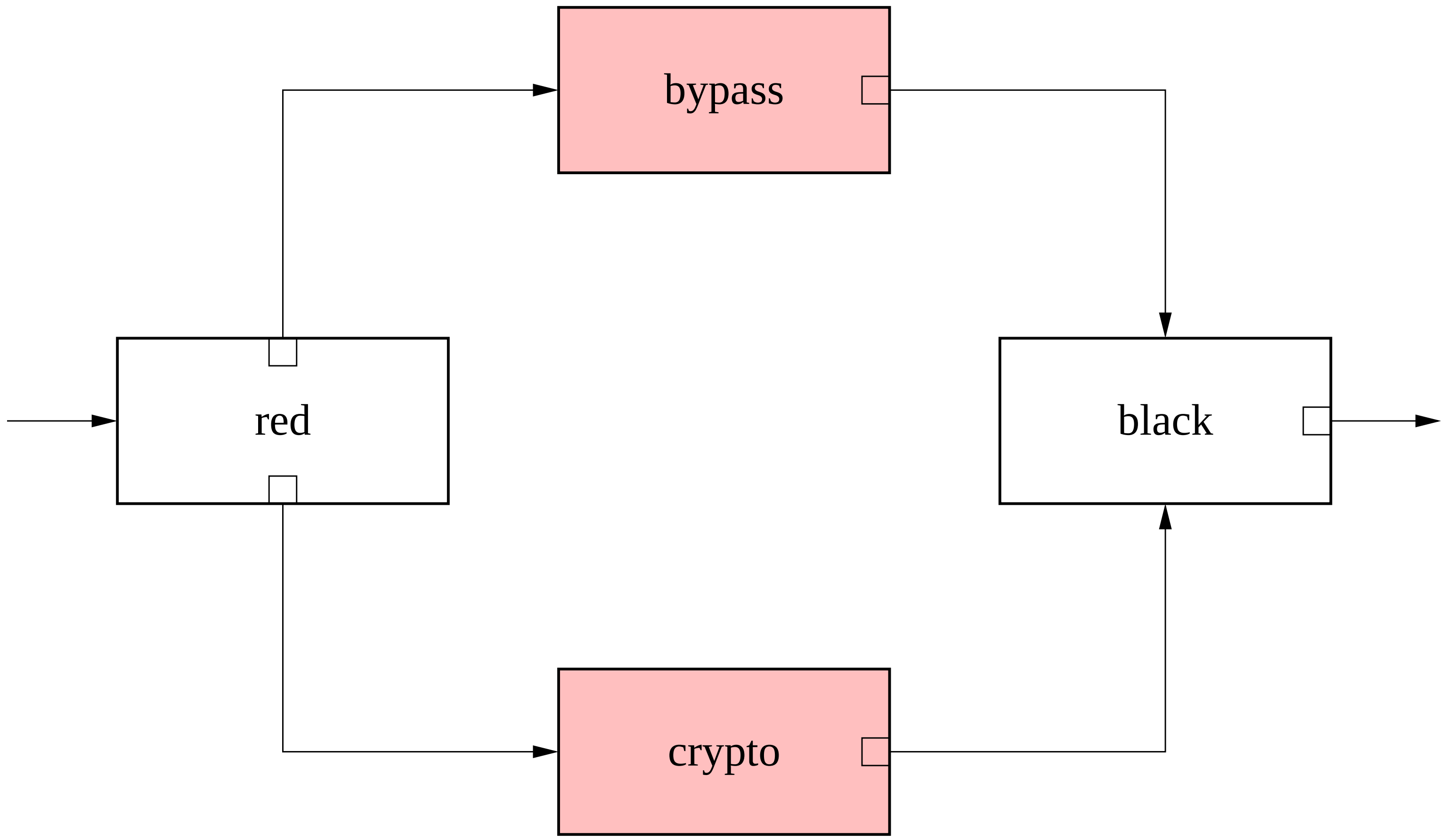
Properties of Arrows

- Need to annotate assumed properties of channels
- Be aware most arrows require a reverse channel

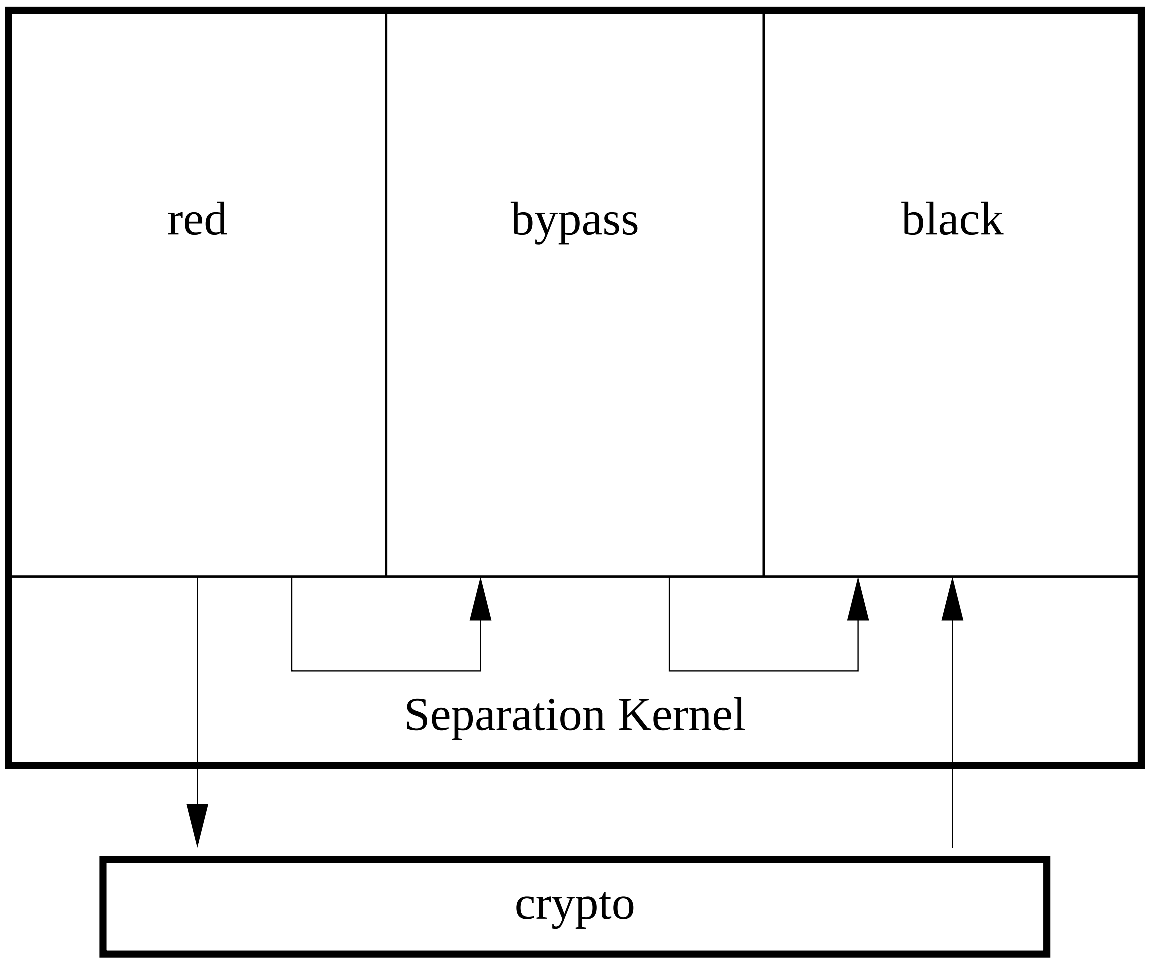


Ports

- Arrows don't contaminate entire state
- Need to identify the explicit interface (or port)



Resource Sharing Level



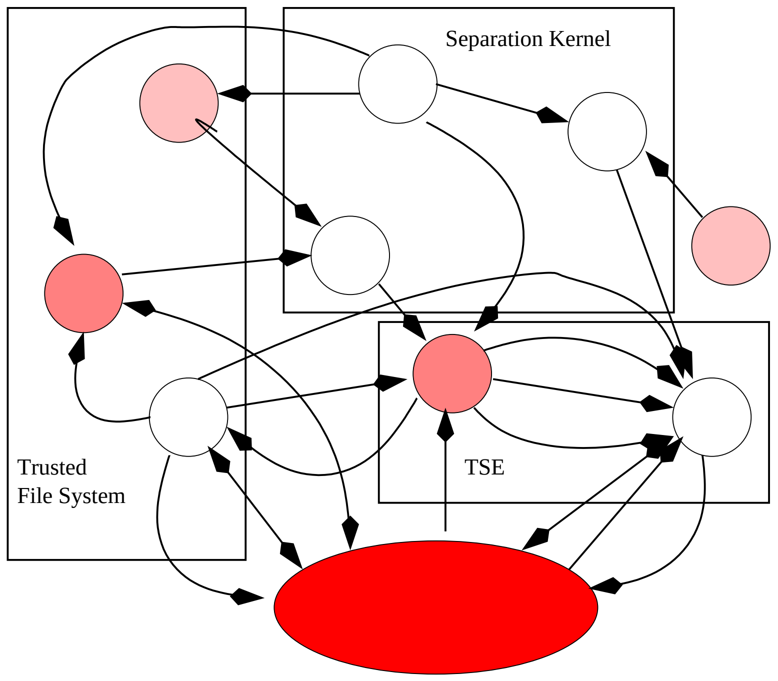
MILS Components

- MILS has a business/uptake model as well as an architectural approach
- Identify generally useful components, fund their protection profiles

  Policy Level: none yet, but could imagine generic guard/filter

  Resource Sharing Level: separation kernels (SKPP), CORBA/Comms (PCSPP), networks (MNSPP), file system, etc.
- Encourage a COTS marketplace to supply compliant components

A Generic MILS System



Care and skill needed to determine which logical components share physical resources (performance, faults)

MILS and HAP

- Many similarities
- Both are two-level approaches
- Difference is MILS lower level develops resource sharing components, HAP provides a specific platform
- Would be best to work together

The MIPP

- Traditionally, MILS is about how decompose systems and allocate components to subsystems
- Urgent need to consider integration
- Need to show that the individual policies of the upper level policy components “add up” to enforce the overall security policy
- And that the lower level resource sharing components do their job, which is to enforce the upper level architectural configuration
  - Invisible to good upper level components

Two Kinds of Components, Two Kinds of PPs

The foundational and operational levels of the MILS architecture have different concerns and are realized by different kinds of components having different kinds of PPs

Operational level: components that provide or enforce application-specific security functionality and policy

- Examples: downgrading, authentication, MLS flow
- Their PPs are concerned with the specific security function that they provide

Foundational level: components that securely share physical resources among logical entities

- Examples: separation kernel, partitioning communication system, console, file system, network stack
- Their PPs are concerned with partitioning/separation/secure sharing

Two Kinds of Components, Three Kinds of Composition

We need to consider three kinds of component compositions

operational/operational: need compositionality

foundational/operational: need composability

foundational/foundational: need additivity

Take these in turn

Compositionality

Operational components combine in a way that ensures **compositionality**

- There's some way to calculate the properties of interacting operational components from the properties of the components (with no need to look inside), e.g.:
  - Component A guarantees P if environment ensures Q
  - Component B guarantees Q if environment ensures P
  - Conclude that $A \,||\, B$ guarantees P and Q
- Assumes components interact only through explicit computational mechanisms (e.g., shared variables)

Composability

Foundational components ensure **composability** of operational components

- Properties of a collection of interacting operational components are preserved when they are placed (suitably) in the environment provided by a collection of foundational components
- Hence foundational components do not get in the way
- And the combination is itself composable
- Hence operational components cannot interfere with each other nor with the foundational ones

Composability makes the world safe for compositional reasoning

Additivity

Foundational components compose with each other **additively**

- e.g., **composable(kernel)** + **composable(network)** provides **composable(kernel + network)**
- There is an asymmetry: partitioning network stacks and file systems and so on run as clients of the partitioning kernel

Going Forward

- Dynamic configuration (cf. Michael's presentation)
- And SOA
- Properties other than/additional to security
  - Cf. partitioning for safety