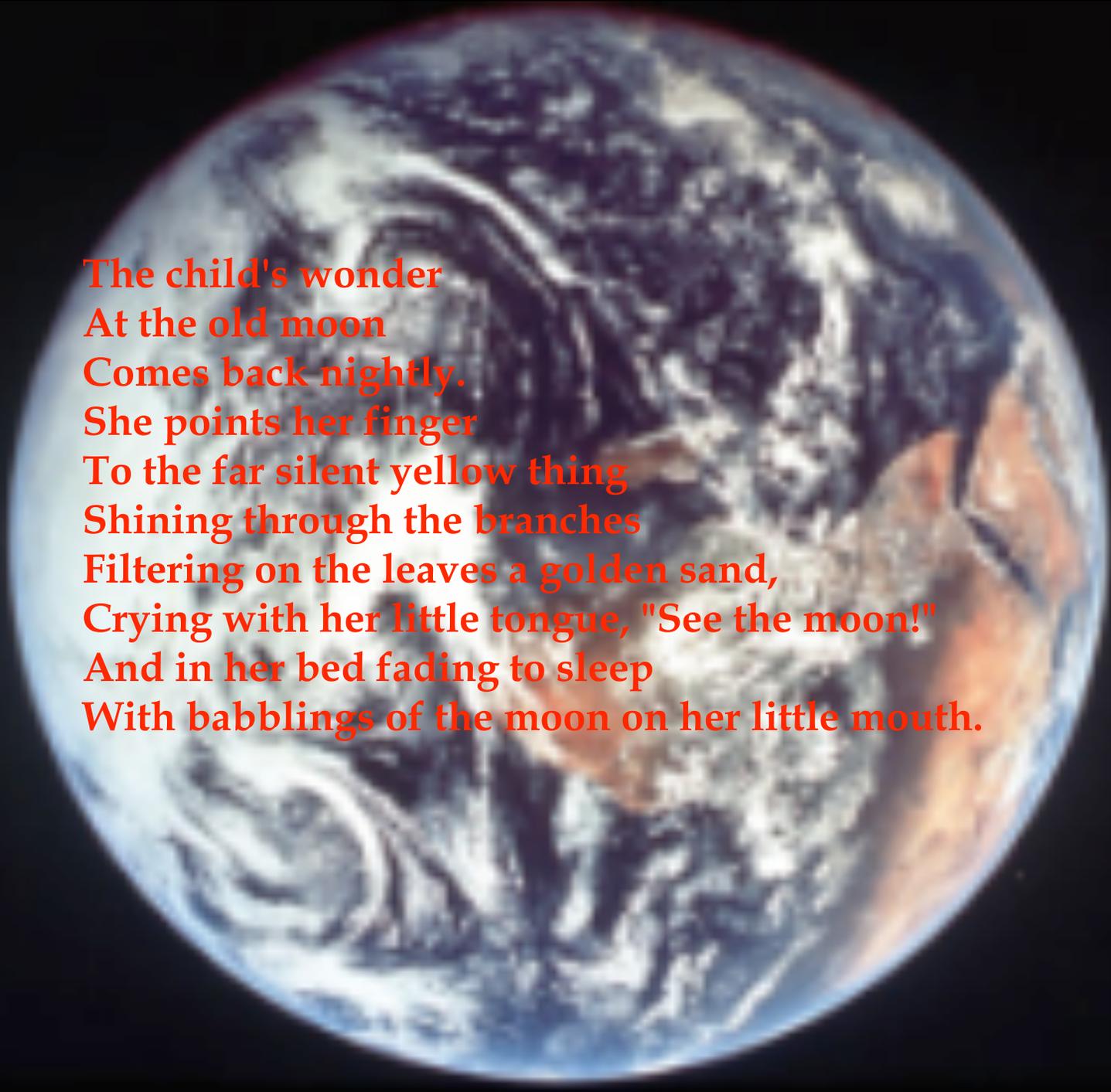


# **Problems, Wise Patterns, and Process**



**Prof. James Coplien**  
**North Central College**  
**University of Manchester**  
**Institute of Science and**  
**Technology**





The child's wonder  
At the old moon  
Comes back nightly.  
She points her finger  
To the far silent yellow thing  
Shining through the branches  
Filtering on the leaves a golden sand,  
Crying with her little tongue, "See the moon!"  
And in her bed fading to sleep  
With babblings of the moon on her little mouth.

# Having lost the dream



*Leaves of poplars pick Japanese prints against the west.  
Moon sand on the canal doubles the changing pictures.  
The moon's good-by ends pictures.  
The west is empty. All else is empty. No moon-talk at all now.  
Only dark listening to dark.*

—Sandberg, 1918

# The deeper dream



- Integrating technology and nature, the universe
- The space program as a benefit to society
- Where is the archive of those ideas?

# A similar plight in software



- Our disciplines are similar:
  - High cost of failure
  - High cost — period
  - In the public eye
  - Traditional reliance of mature technology
  - A century of technology to build on (Bell, Goddard)
- But today the basics are fading

# Technological Science and Natural Science



- Technological:
  - Humankind in control
  - Preformed parts — the Cartesian legacy
- Natural
  - Harmonizing with nature
  - Wholes
  - Repair
- These approaches offer insight into design

# The Architect Alexander



*But it is impossible to form anything which has the character of nature by adding preformed parts.*

When parts are modular and made before the whole, by definition then, they are identical, and it is impossible for every part to be unique, according to its position in the whole.  
(Alexander, 1979: p. 368)



# Indian Hill. 1963



# Indian Hill. 1976



# Indian Hill, 1988



# What is a pattern?



## Center of Gravity

...there is an inherent struggle between the center of pressure and the center of gravity in a high-speed missile. Once the PAYLOAD SPACE and FUEL LOADING are arranged, one must engineer the rocket for balance.

\* \* \*

**The pressure on the fins and the mass of the engine compete for control, each exerting leverage on the rocket direction.** Most of the rocket's mass is in the rear, with lighter payload frequently residing in front. This problem is aggravated in multi-stage solid-fuel configurations, since it moves the center of gravity even further toward the back.

You can always over-engineer the fins to guarantee stability, but the drag reduces the apogee.

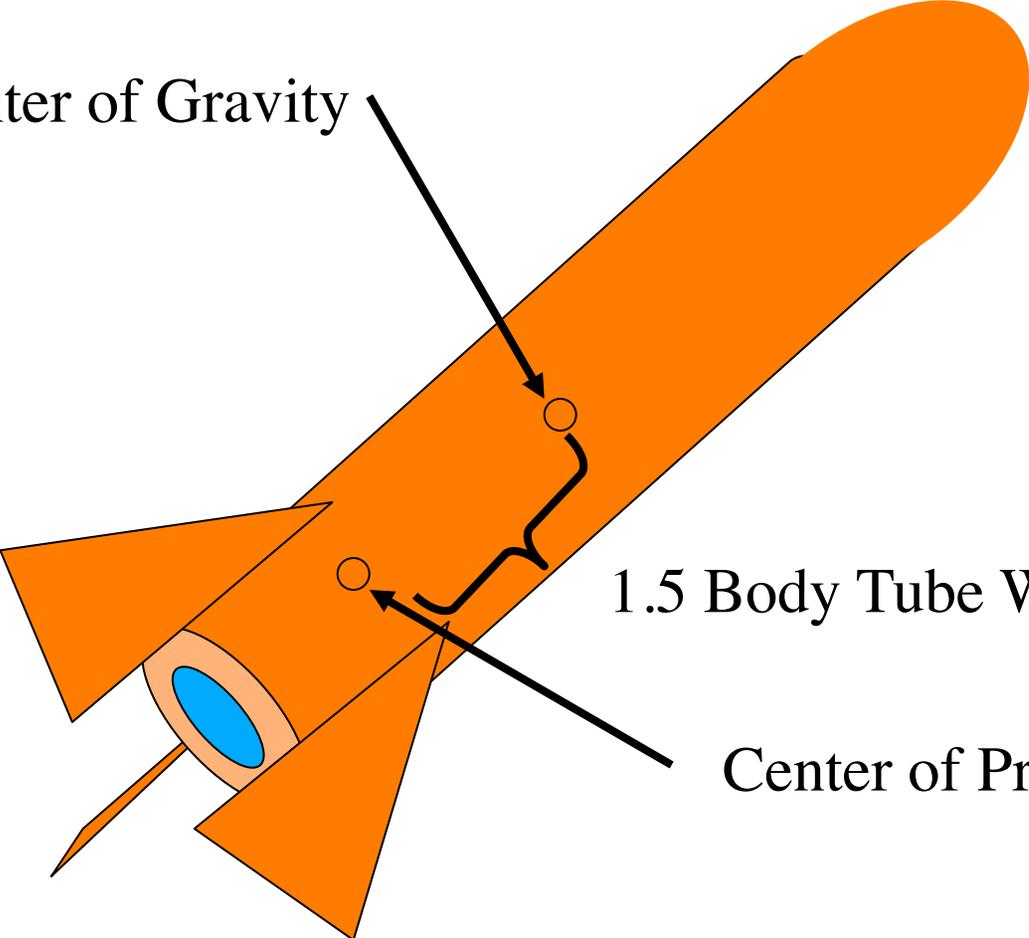
Therefore:

**Locate the center of pressure one-and-a-half body tube widths behind the center of gravity.** This provides enough aerodynamic correction at speed to keep the missile from "falling over", and it keeps going in the same direction.

\* \* \*

A good center of gravity will add stability which not only ensures the rocket will go in the intended direction, but that it will fly with less wobble and achieve higher performance. Adjust CENTER OF GRAVITY with NOSE WEIGHT, potentially elongating the body tube, and TRAILING FIN to move the center of pressure back.

Center of Gravity



1.5 Body Tube Widths

Center of Pressure

# How does a pattern work?



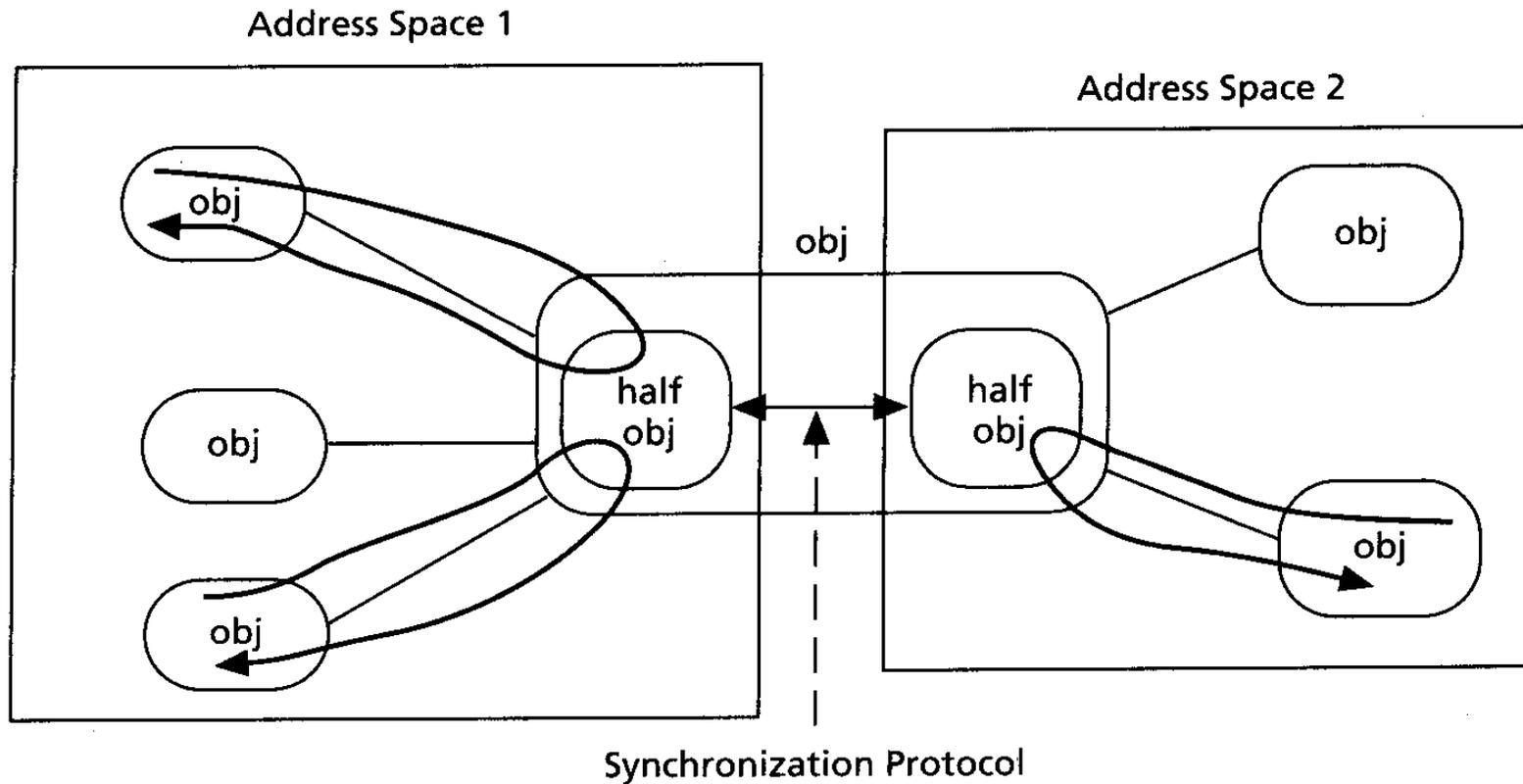
- One at a time, compose patterns
- Tailor each pattern to what has been built before
- *A process* of continuous adaptation
- *Piecemeal growth* to build systems
- *Local Repair* to add structure
- A system is always in repair

# What's the idea?



- Marketed as knowledge capture
- Patterns compose to generate systems
- The patterns that generate systems are called a *pattern language*
- A pattern provides context for other patterns
- Each pattern unfolds in the context where it is applied

# Half-Object Plus Protocol



# THREE PART CALL PROCESSING (ADD A SWITCH)

**Problem:** Efficient interfacing between many signaling types.

**Context:** Toll switching where each switching center must communicate with many various signaling types.

**Forces:** Want flexibility to interwork between any signaling types.

More signaling types are being developed constantly.

There are some common functions that are signaling type independent.

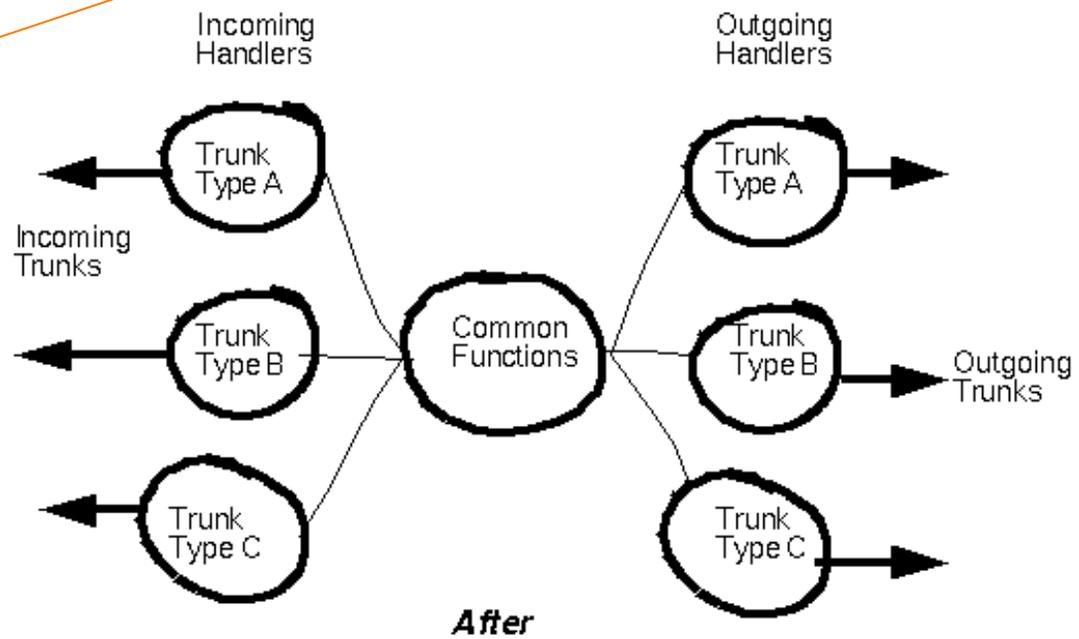
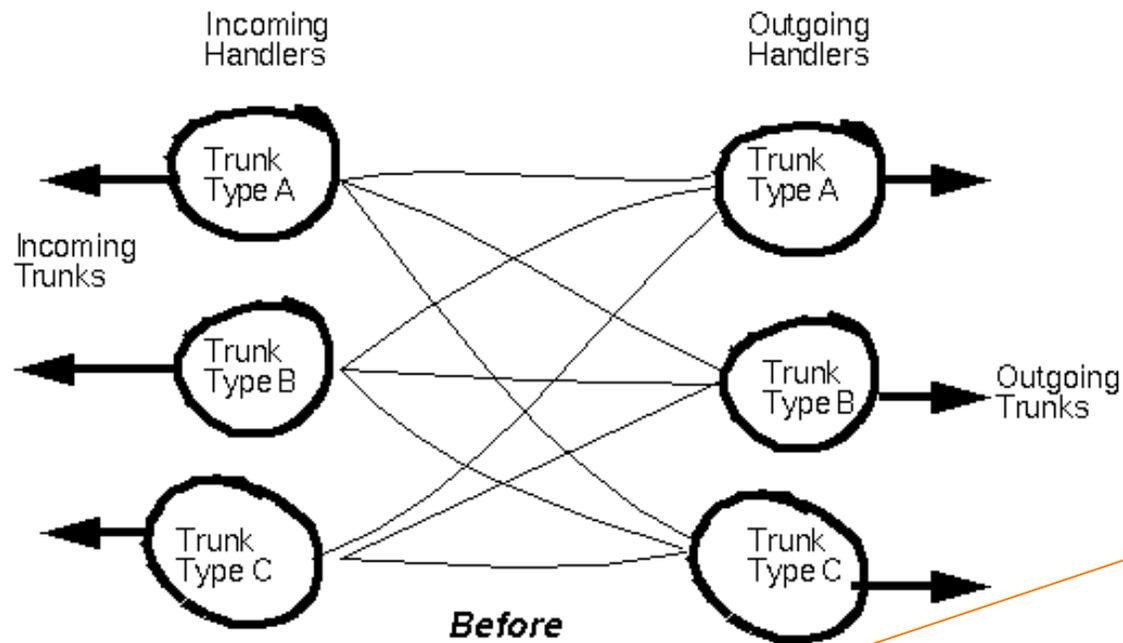
It is expensive to make changes to every existing signaling type to add new functions.

Want centralization and standardization, for ease of maintenance and to facilitate understanding.

**Solution:** Distribute call processing into something that can be processed by three separate pieces: Incoming trunk handling, Outgoing trunk handling, and non-trunk specific (common) pieces.

**Resulting Context:** The parts of the software that know about signaling types only need to know about the ones that they process. They have been isolated from all other signaling types.

Non signaling specific types of functions are concentrated into separate modules for ease of maintenance.



# THREE-PART CALL PROCESSING, continued



**Rationale:** Easily extensible. To add new signaling types, add the trunk handlers and make minor changes to some common functions.

This pattern applies for the same reason that we have telephone switches—the complexity of direct connections between each node in a network.

**Author:** Robert Hanmer, 5/9/1995

**Reference:** P. D. Carestia, F. S. Hudson. *No. 4 ESS: Evolution of the Software Structure*, BSTJ, Vol 60, No. 6, Part 2

# LEAKY BUCKET COUNTERS

Pattern: **Leaky bucket counters**

Problem: **How do you deal with transient faults?**

Context: **1A/1B processor 4ESS. As stores (dynamic RAM) got weak, we'd have a store taking a trap refresh (parity) failure. This is applicable both to 1A dynamic RAM and 1B static RAM.**

Forces: **You want a hardware module to exhibit hard failures before taking drastic action. Some failures come from the environment, and should not be blamed on the device.**

Solution: **We count faults or events (usually faults) and decrement the count on a periodic basis to deal with transient faults. There are different leak rates for different 1A subsystems: a half-hour for the store subsystem; other subsystems include the interface bus, CC, etc. We developed a strategy for 1A dynamic RAM. On the first fault in a store (within the timing window), we'd take the store out of service (OOS), diagnose it, and then automatically restore it to service (if it failed -- but usually they never failed diagnostics at this point). On the second, third, and fourth failure (within the window), you just leave it in service. For the fifth episode within the timing window, take the unit OOS, diagnose it, and leave it out.**

Resulting Context: **A system where errors are isolated and handled (by taking devices OOS), but where transient errors (e.g., room humidity) don't cause unnecessary OOS action.**

# LEAKY BUCKET COUNTERS, continued

Rationale: Goes into all our software. The history is instructive: in old call stores, why did we collect data? For old call stores, the field replacement unit (FRU) was a circuit pack, while the failure group (FG) was a store comprising 12 or 13 packs. We needed to determine which pack is bad. Memory may be spread across 7 circuit packs; the transient bit was only one bit, not enough to isolate the failure. By recording data from four events, we were better able to pinpoint (90% accuracy) which pack was bad, so craft didn't have to change 7 packs. Why go five failures before taking a unit OOS? By collecting data about failure on second, third, and fourth time, you are making sure you know the characteristics of the error and are reducing the uncertainty about the FRU. By the fifth time, you know it's sick and need to take it OOS. Decreasing the count on the store one per half hour creates a sliding time window. If count clears (goes to zero), the store is considered fine at that point. Humidity, heat, and other environmental problems could cause transient errors which should be treated differently (i.e., pulling the card does no good).

See, for example, Fool Me Once.

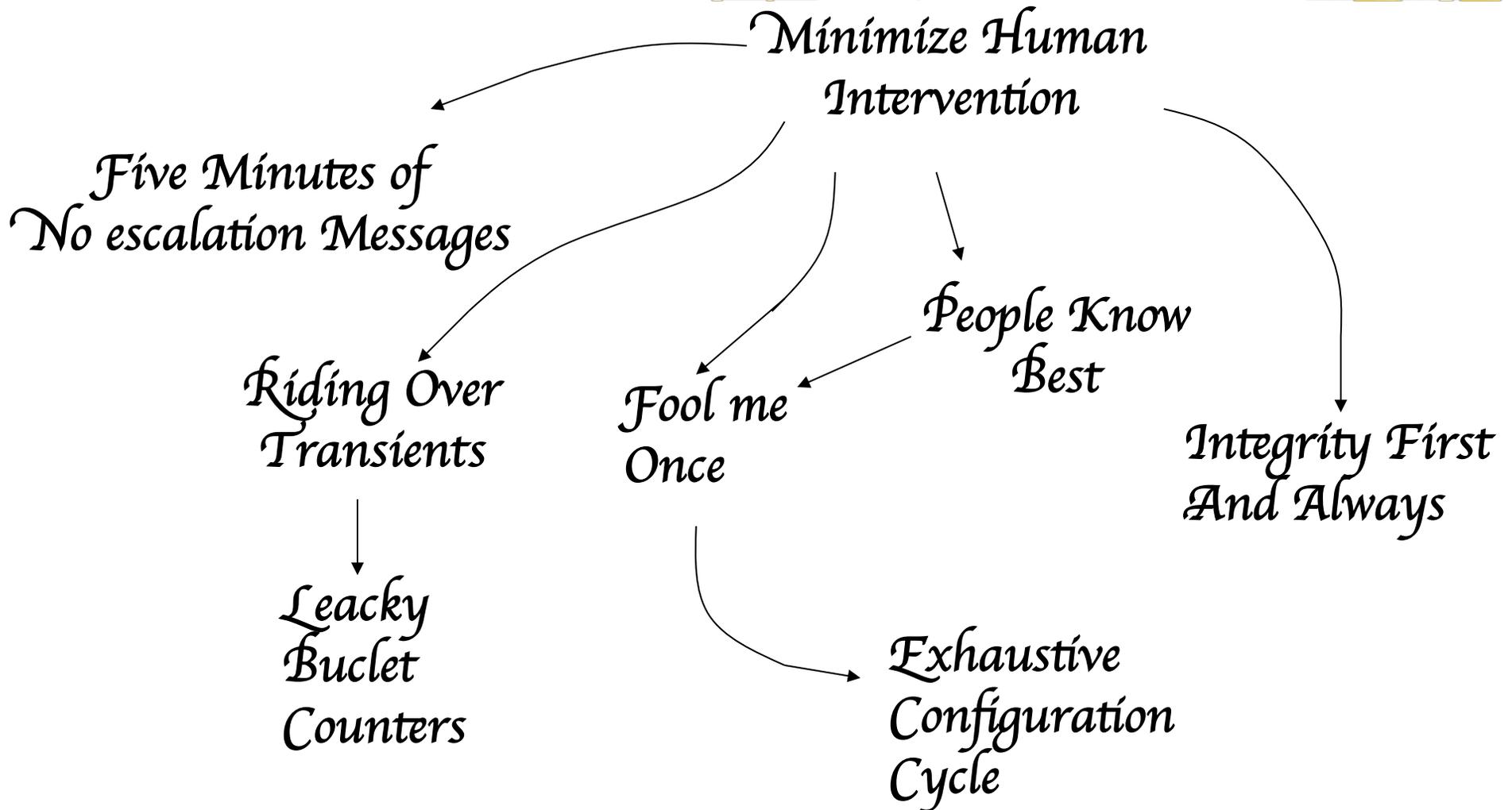
Uses pattern Riding Over Transients.



*As an element in the world, each pattern is a relationship between a certain context, a certain system of forces... and a certain **spatial** configuration which allows these forces to resolve themselves.*

*As an element of **language**, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant. — TTWOB, 247*

# A Small Pattern Language



# What, then, is problem-solving?



- Guided by experience and metaphor
- Generativity: building systems that solve their own problems
  - High-reliability as contrasted with fault tolerance
  - The human as part of the system

# Patterns and the future of our disciplines



- The human component is ever-present
  - We must value the past
  - We should try to interpret current problems—and successes—in terms of the past
  - We should leave a legacy of knowledge, and a legacy of beauty, for our children
- 