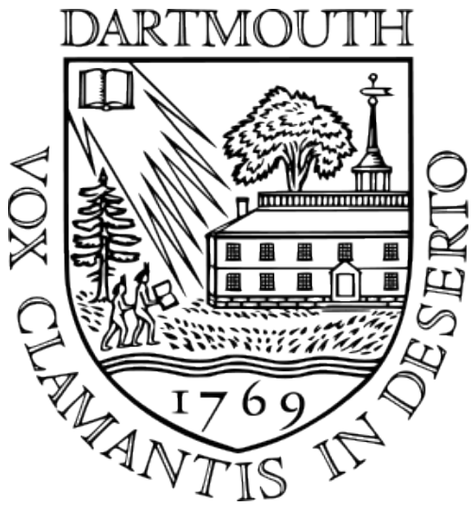
Intent Semantics in the ABI

Sergey Bratus, Julian Bangert



Outline

- From faulty classic policies to a new sweet spot
- ABI-level objects and security policy
- ABI-level policy examples
- Why this works on x86
- Future directions

Traditional Security

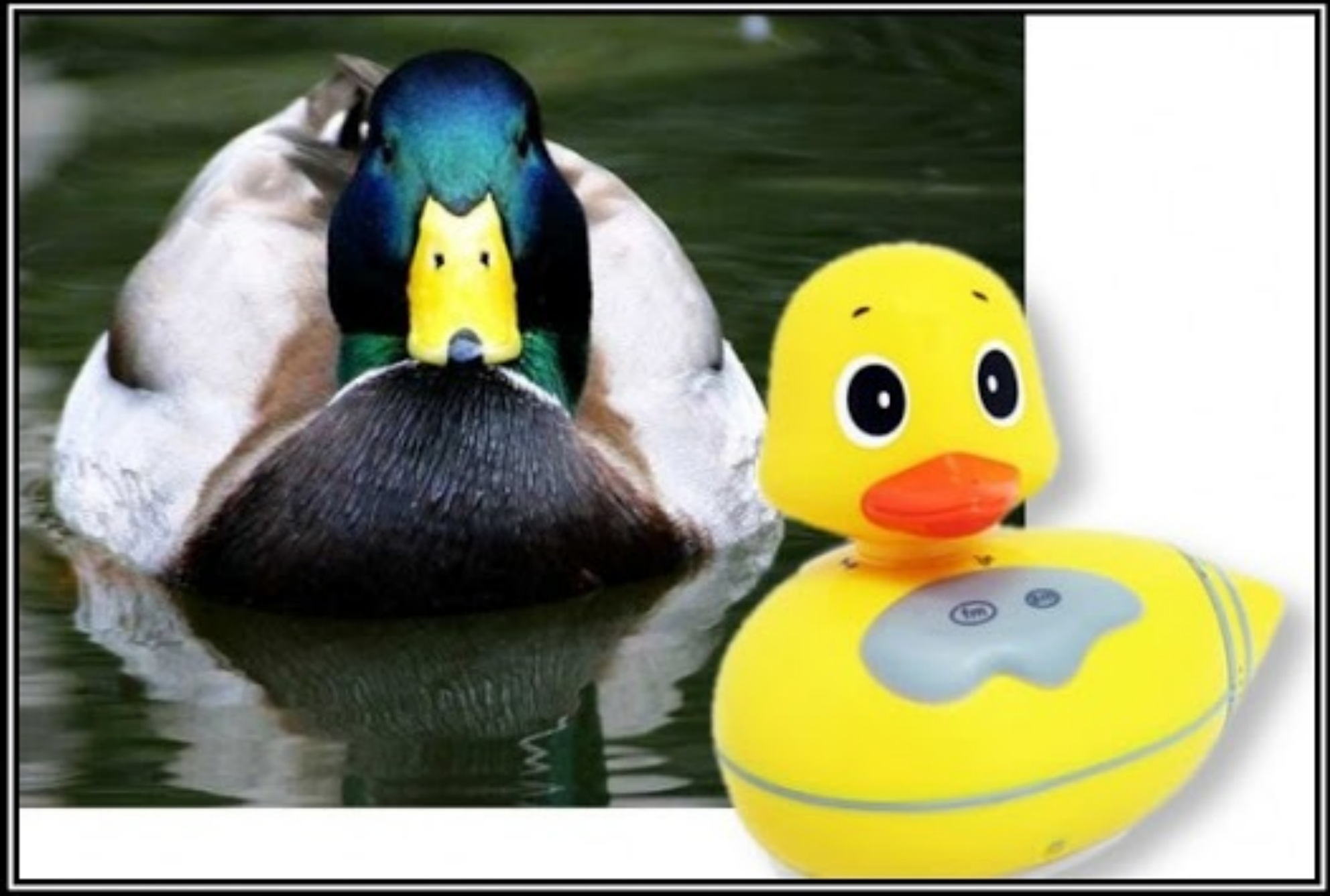
- Traditional security models assume:
  - One process does one thing
  - Static bag of permissions for the entire process
  - Usable at any point, in any order, any number of times

If JS is your OS, what is your reference monitor?

- Is your data in objects you can **label**?
  - Does it even touch any filesystem?
- If it is, can you **trap** on access to it?
  - Does it ever go through a syscall or VM lookup?
- For DOM: Is Same Origin even the right labelling scheme?

A process is a process is a process

- For a "**task**", the "bag of permissions" model is adequate. For a "**process**", it isn't
- A "process" goes through **changes** over time
- Yet in policy we treat it as just a "task", monolithic
- This is wrong and counter-intuitive
- What are the "**units**" or "**phases**" of a process?

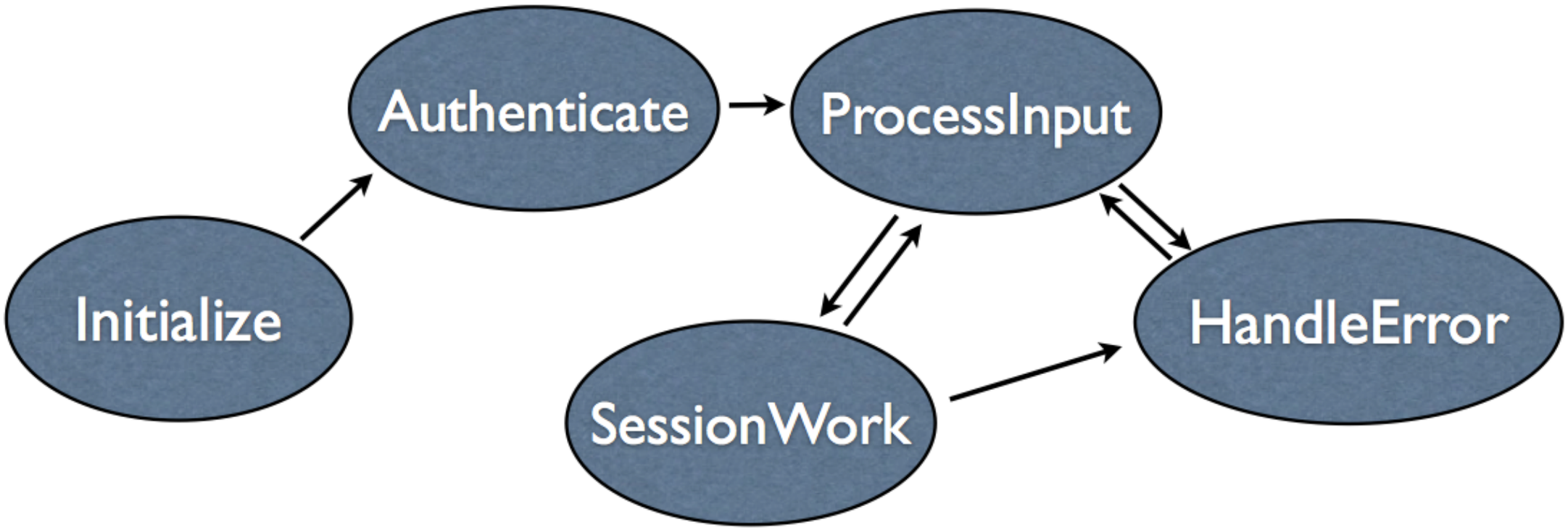


LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

http://www.tomdalling.com/blog/software-design/solid-class-design-the-liskov-substitution-principle/

Process phases



- "Phase" ~ code unit ~ EIP range ~ memory section

"Some thoughts on security after ten years of qmail", D.J. Bernstein, 2007

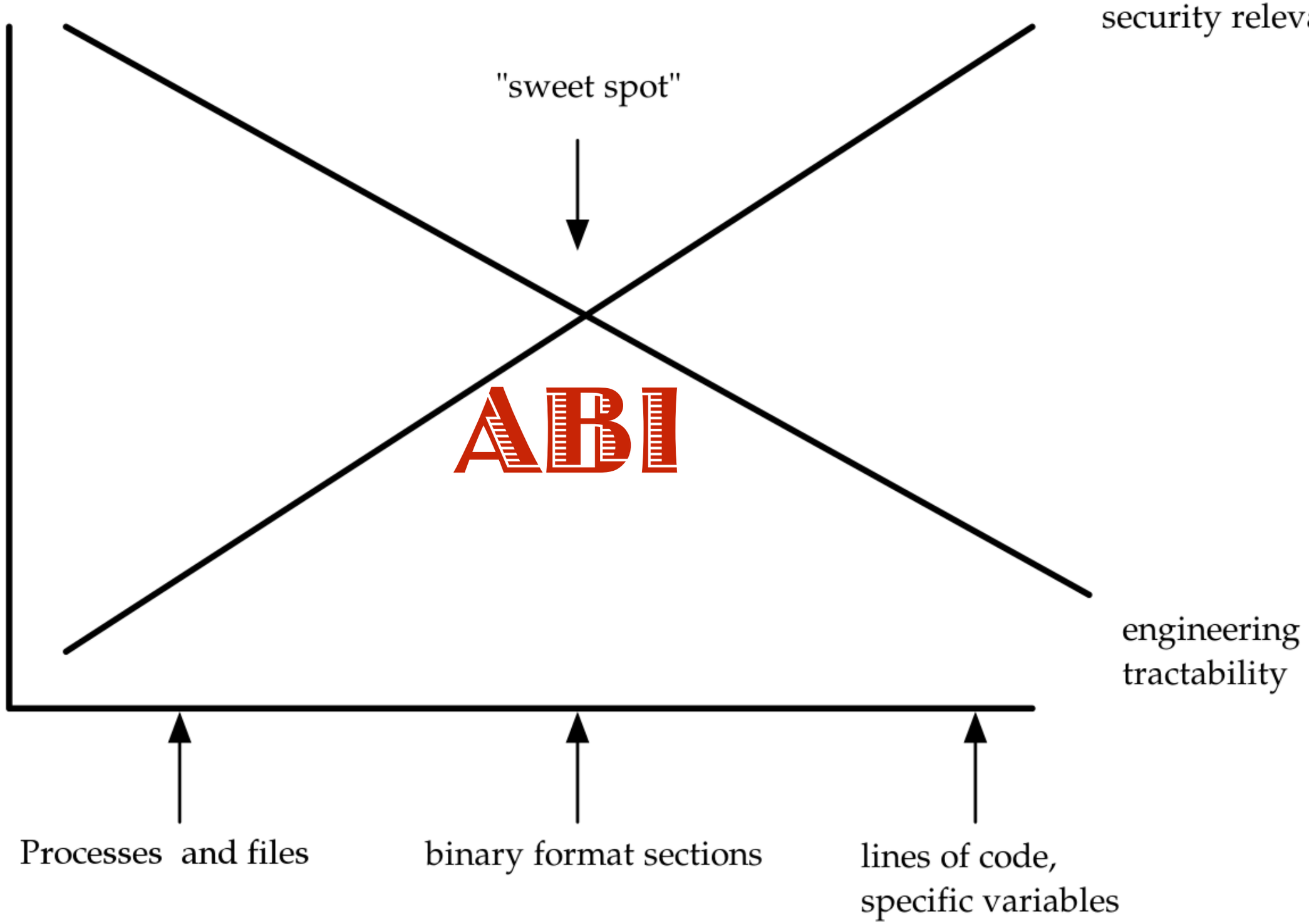
- Used process isolation as security boundaries
  - Split functionality into many per-process pieces
- Enforced **explicit data flow** via process isolation
- Avoided in-process parsing
- Least privilege was a distraction, but **isolation** worked

http://cr.yp.to/qmail/qmailsec-20071101.pdf

Traditional Security vs. Modern Software

- Software is complicated, integrates many functions
  - "The *** Shopping App Now Backs Up Your Photos"
- High engineering costs to manually isolate **components**/functional units a-la qmail
- Semantic subdivision occurs at **ABI section** level
  - Code & data sections reflect different **intent**
  - Functional units ~ ABI semantic units

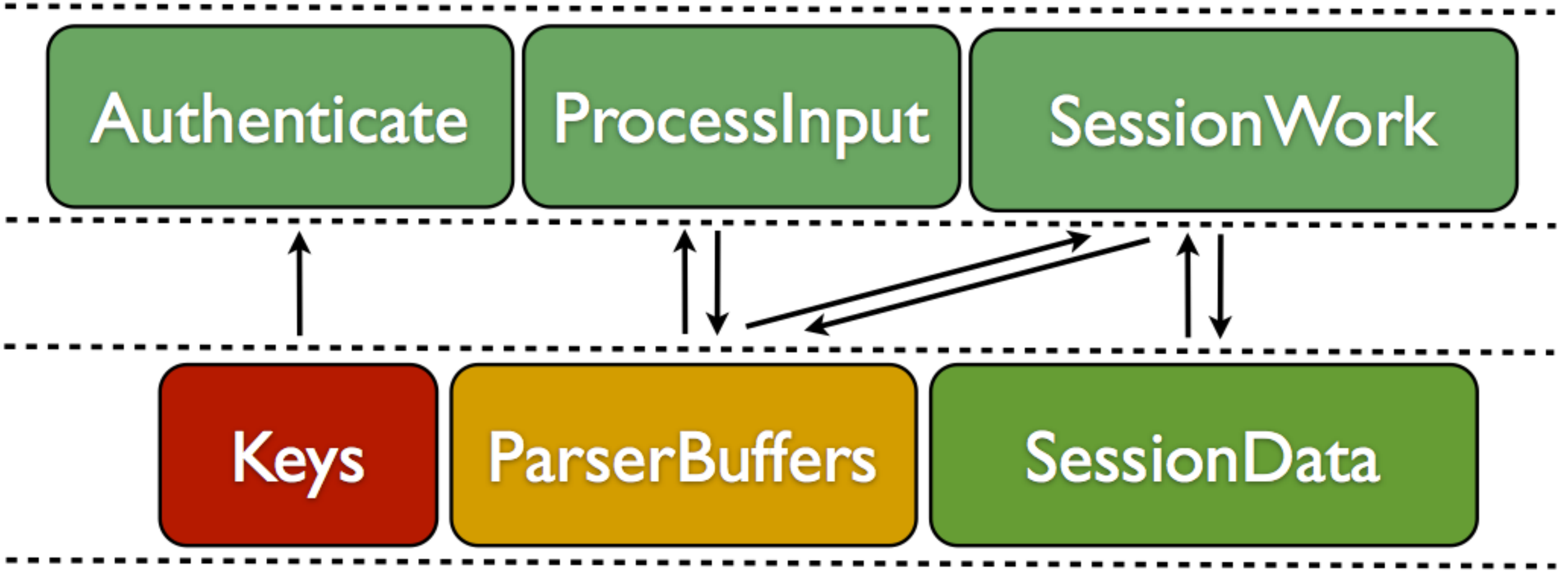
Policy Granularity: ABI is the Sweet Spot



Intent-level semantics

- "*The gostak distims the doshes*"
  -- Andrew Ingraham, 1903
  - Non-dictionary words, English grammar
  - Semantics == relationships between terms
- **Relationships** between code & data sections reflect their **intent**, often uniquely

Access relationships are key to programmer intent



- Unit semantics ~ Explicit data flows (cf. *qmail*)

Separation of concerns in OS engineering practice

- Sections describe the **intent** of code and data
- Example: Dynamic linker/loader operates on
  - **GOT** in ELF, function stubs in **PLT**
  - **IAT**, import & export data tables in PE

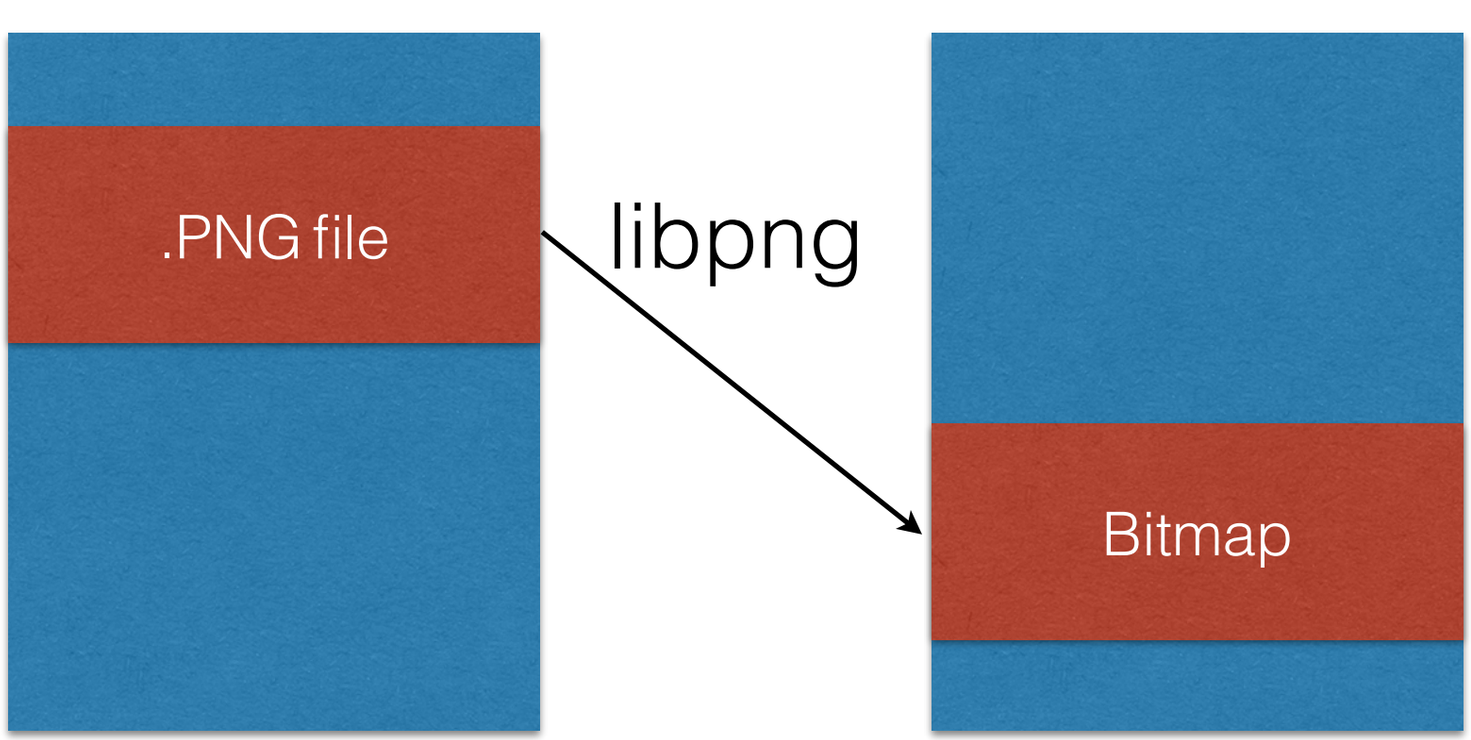
Enforcing

- Modern OS loaders **discard** section information
- New architecture:
  - '**Unforgetful** loader' preserves section identity after loading
  - Enforcement scheme for **intent-level semantics**
  - Better tools to capture semantics in ABI

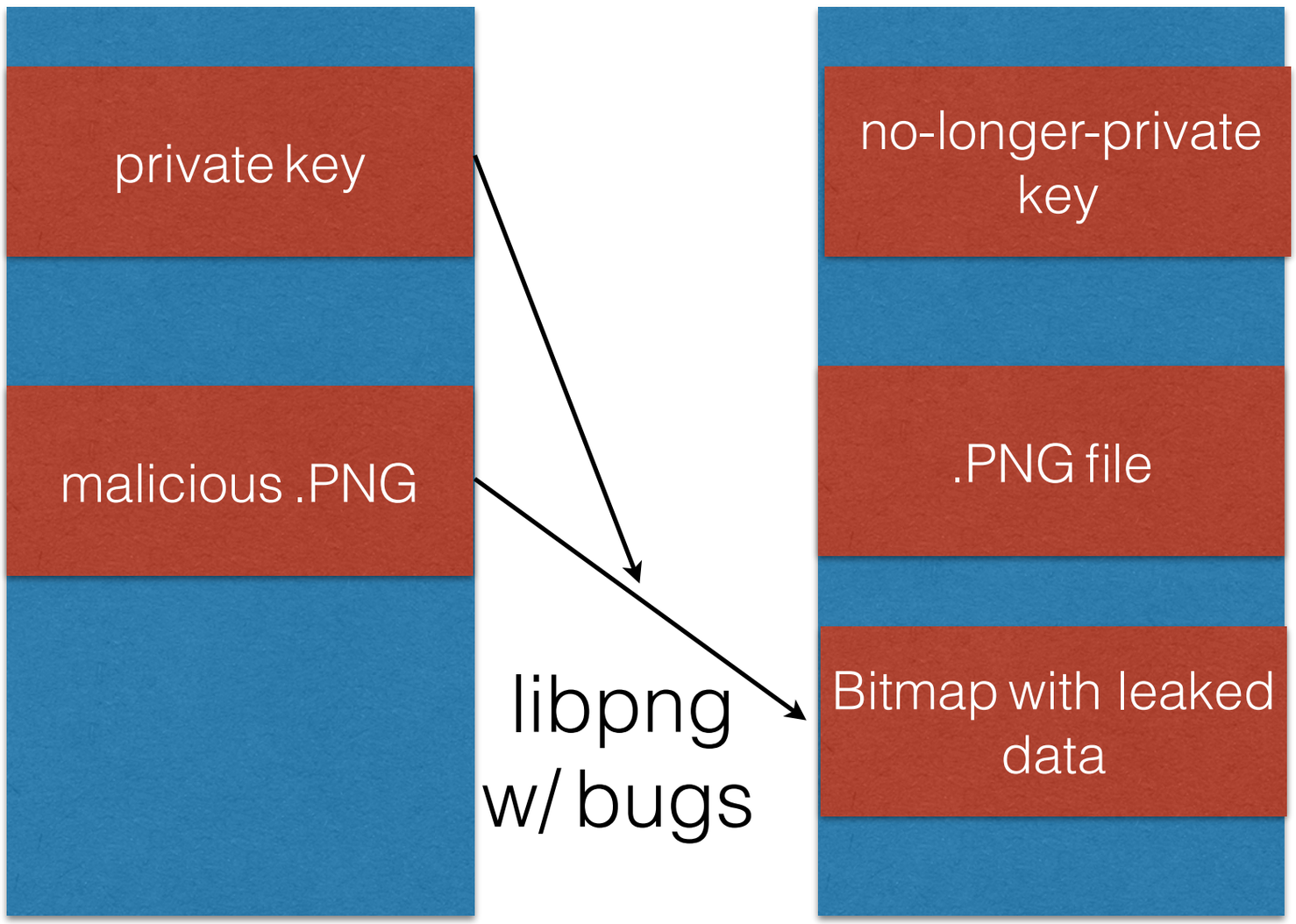
Motivating Example

Example policies

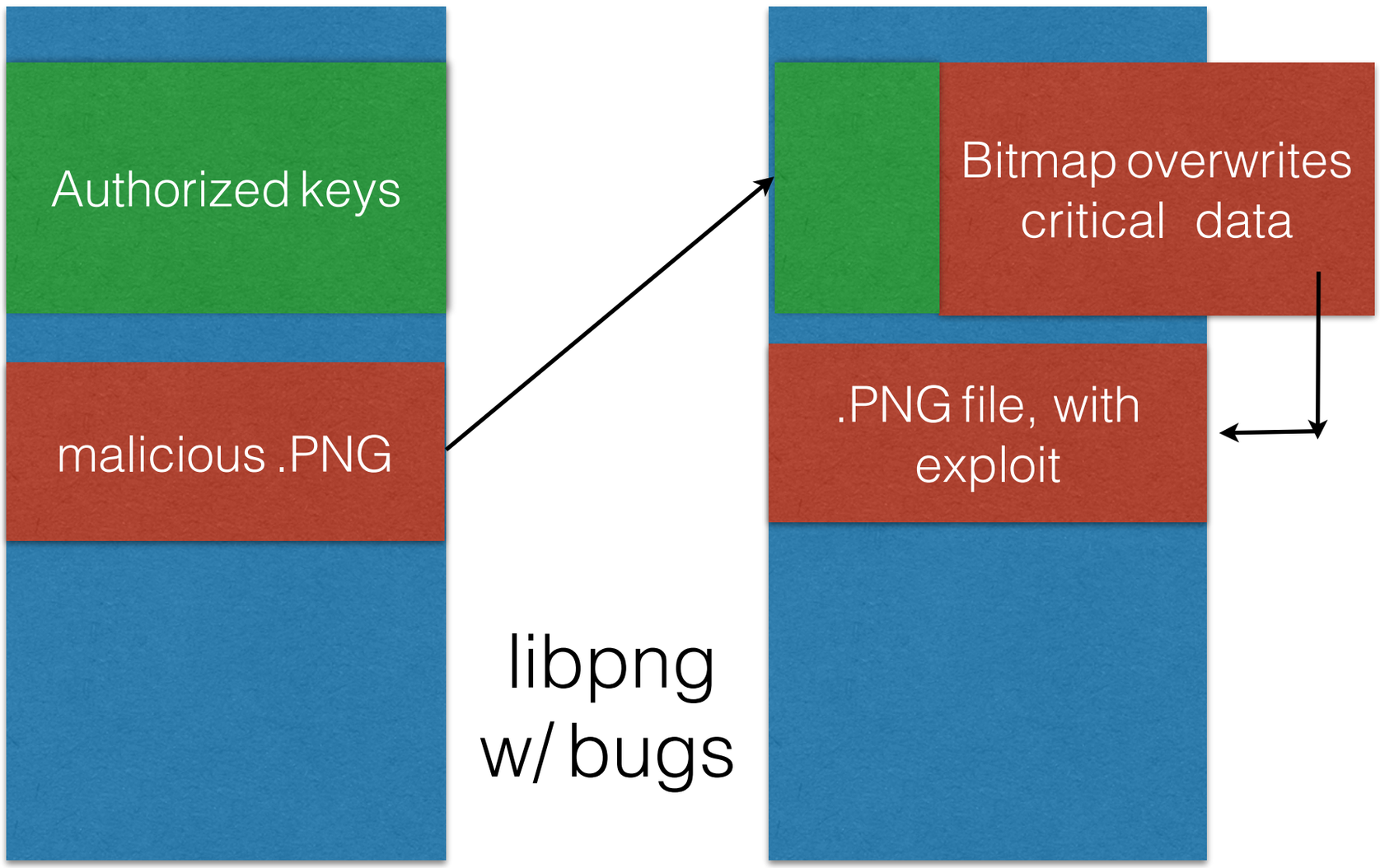
- Web application decompresses a PNG file
- Mental model



What attackers see

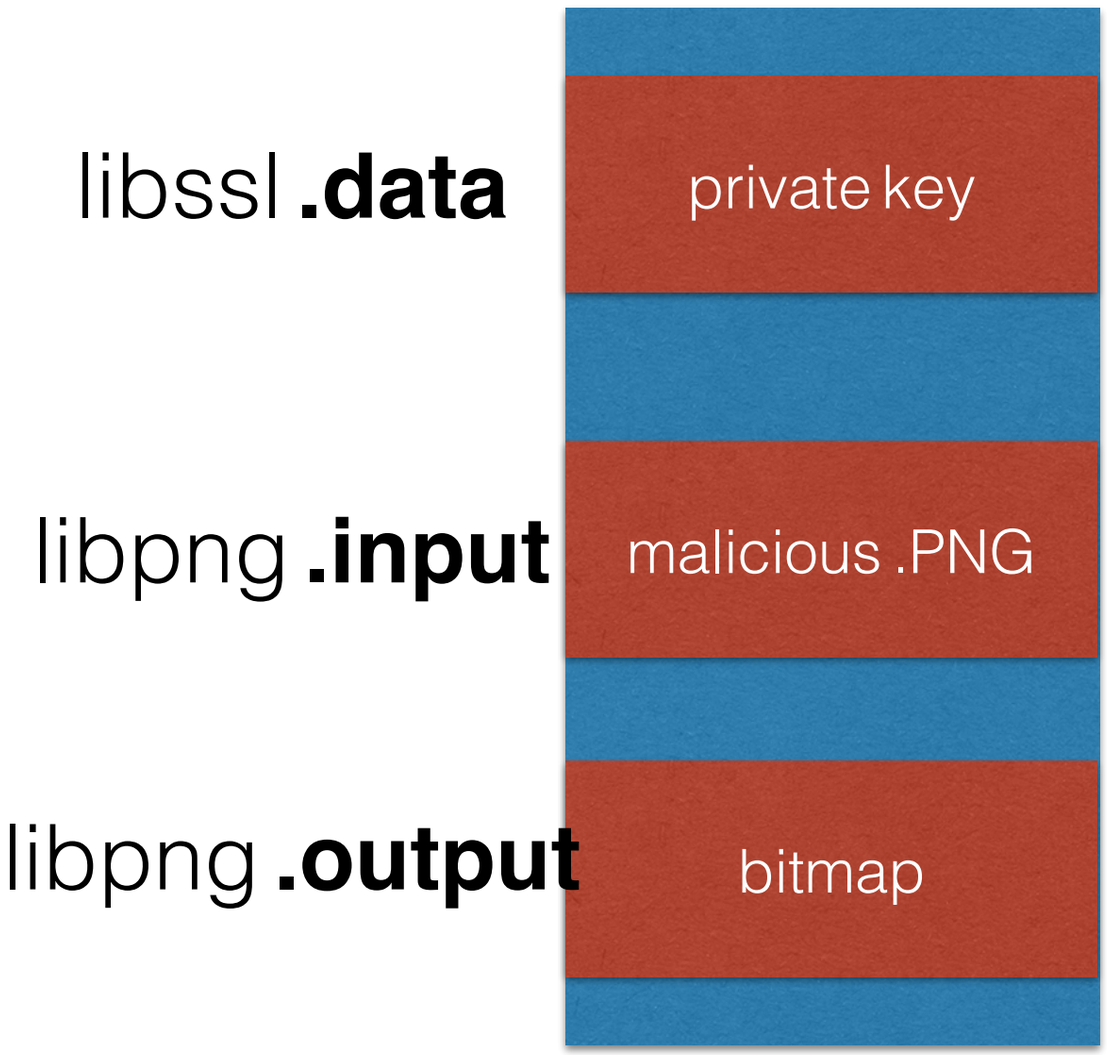


Or


Authorized keys
malicious .PNG
libpng
w/ bugs
Bitmap overwrites
critical data
.PNG file, with
exploit

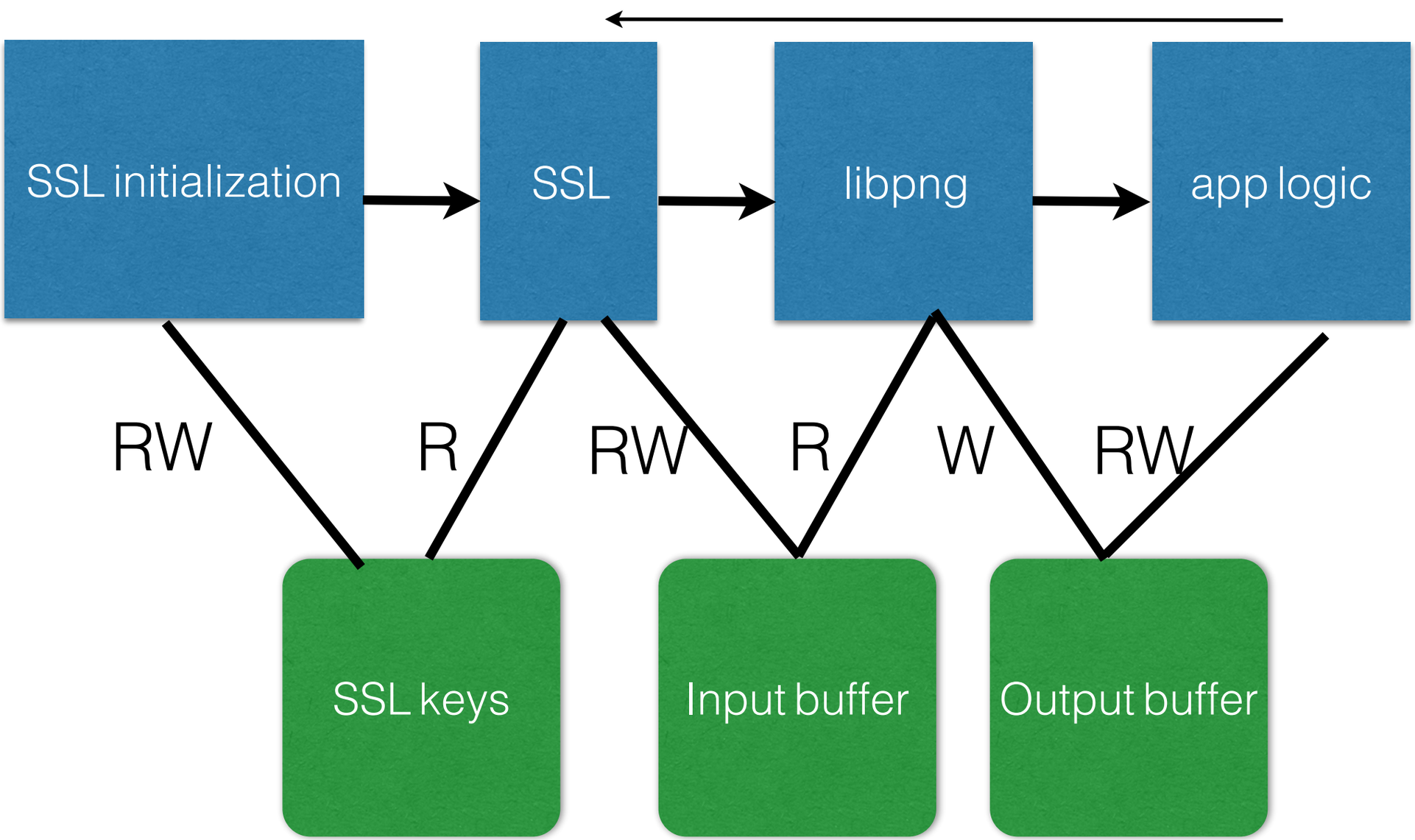
Mapping it into the ABI

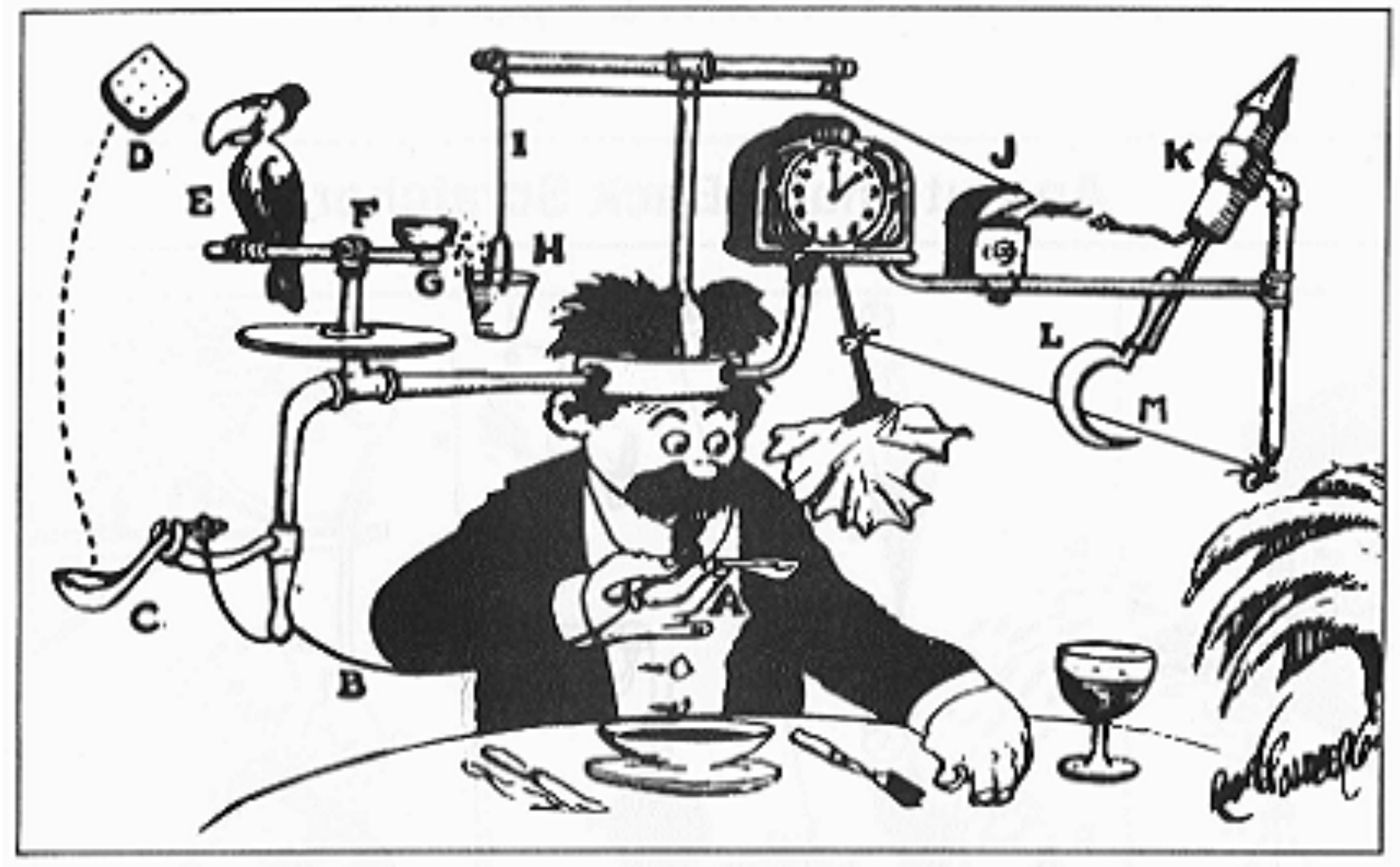
- Easy to introduce new sections
- Each code segment can get different permissions



- Only libssl.text can access libssl.data
- libpng.text can only access libpng.input and libpng.output
- And libpng.input can only be read by libpng.

Back to our example





The Implementation

http://upload.wikimedia.org/wikipedia/commons/a/a6/Professor_Lucifer_Butts.gif

Implementation on X86

- Prototype on Linux with X86 virtual memory
- Each **state** of execution sees a different **subset of the address space**
- **Traps** handle state transitions by changing *CR3*
- Each state has its own **page tables** that cache part of the address space, reusing existing TLB invalidation primitives.
- Use **PCID** on newer processors to reduce TLB misses

Prototype:
Cloud to the rescue!

- Performance hit still rather bad: 30% on simple NGINX benchmark isolating all libraries
  - Too many state transitions on the hot path
  - Policy must be adapted to application structure
- Less overhead (~15%) when running on KVM
  - KVM already incurs performance costs, so we don't have to suffer them
  - KVM also optimizes virtual memory handling

Binary Rewriting Tools

- Policy injection through metadata rewriting:
  - ***Mithril***, currently only implemented for ELF
    - Translates binaries into a *canonical form* that is less context-dependent and can be easily modified
    - Tested on the **entire** Debian x86_64 archive, producing a bootable system
      - ~25GB of packages

Future directions

- Working on enforcing ELFBac-style policies with CFI
- Implementation to ARM (because phones rule!):
  - Domain Control Register: 16 sub-spaces that can be disabled/enabled without flushing caches
  - Can handle a sub-lattice of an ELFbac policy to reduce supervisor entries.
  - Would have to run all user space under virtualization, in kernel mode

Takeaway

- Per-process bags of permission are no longer a suitable basis for policy
- Instead, ABI-level memory objects at process runtime are the sweet spot for security policy
- Modern ABIs provide enough granularity to capture programmers intent w.r.t. code and data units
  - Intent-level semantics compatible with ABI, standard build/binary tool chains