1) Buffer overflows
2) Baggy bounds checking
3) Integer overflows and KINT
4) Privilege separation and OKWS
6) OS Isolation and Capsicum
7) Sandboxing and Native Client
8) Networking and TCP/IP security
9) Network protocols and Kerberos
10) Web security, XSS, CSRF, Injection
11) Web apps, frameworks, Django
12) SSL, HTTPS, and ForceHTTPS

Least Privilege Principle

privilege separation, OS isolation, sandboxing

Sergio Benitez

Outline

- Least Privilege Principle
  - Only give as much permission as needed
- Privilege Separation: OKWS
- OS Isolation: Capsicum
- Code Sandboxing: Native Client

Privilege Separation

- Independent functionalities...
  - access independent data
  - act as independent users
  - have independent permissions
  - explicitly communicate

OKWS: OKCupidWS

- Web dev. libraries and helper processes
- (Tries to) follow(s) least privilege principle
- Aims to implement privilege separation
  - Using standard Unix stuff
- ZookWS follows it almost directly

Privilege Separation

- Independent functionalities...
  - access independent data
  - act as independent users
  - have independent permissions
  - explicitly communicate

OKWS

- Independent functionalities...
  - access independent data (*chroot*)
  - act as independent users (*setuid/setgid*)
  - have independent permissions (*chmod*)
  - explicitly communicate (*rpc*)
  - are independent processes (*fork*)

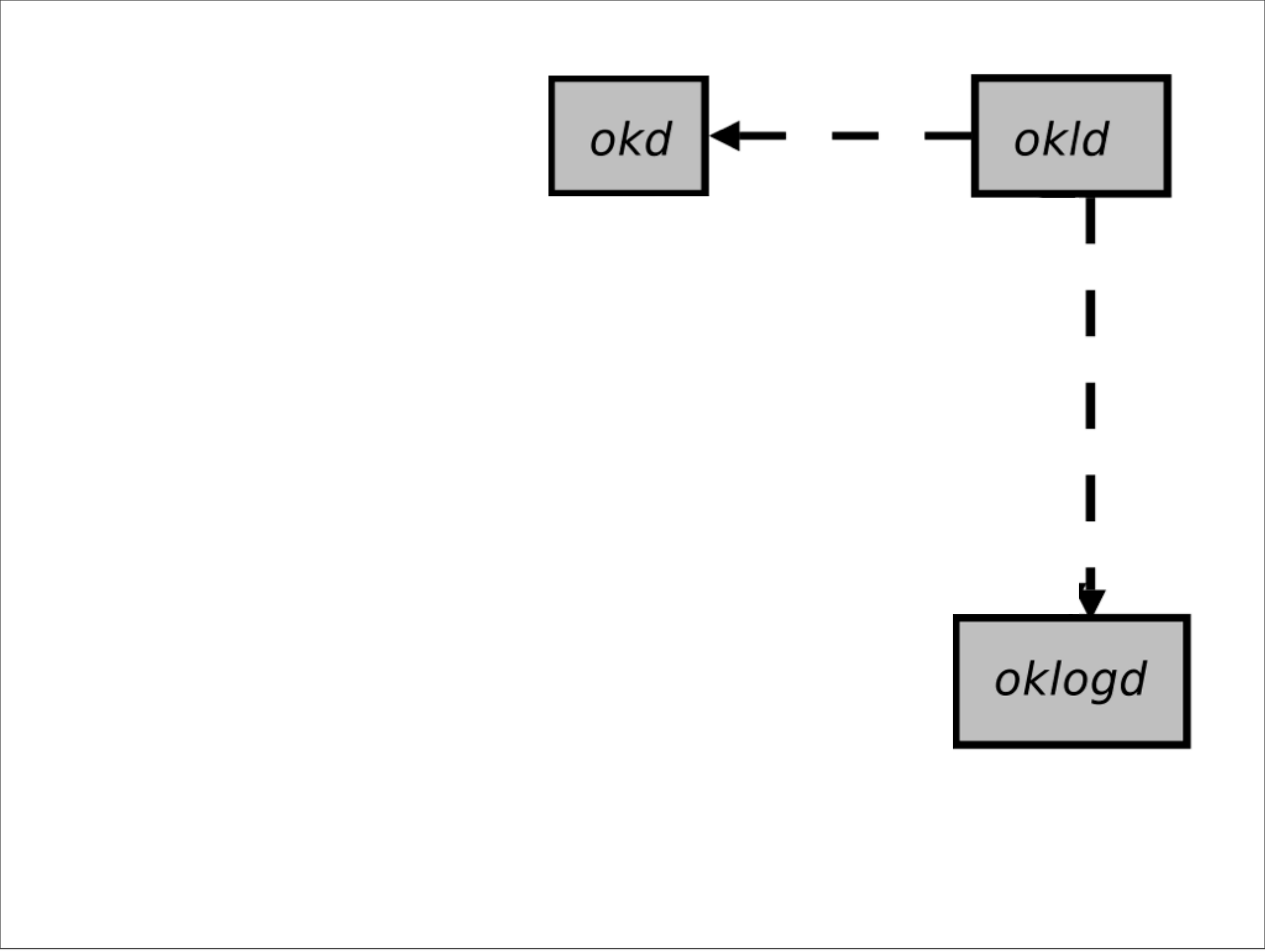
OKWS Design

- OKLD: OK Launcher Daemon
  - Runs as root, starts all processes, services
- OKD: OK Dispatcher
  - Demultiplexes HTTP requests to services
- PubD, OKLogD: Pub/Logger Daemons

Divided into four processes. Only OKLD runs as root – it starts all other services. OKD receives all HTTP requests, figures out where to route them, and then sends them along to the proper service. PubD manages static items, like HTML templates, and LogD lets services log things.

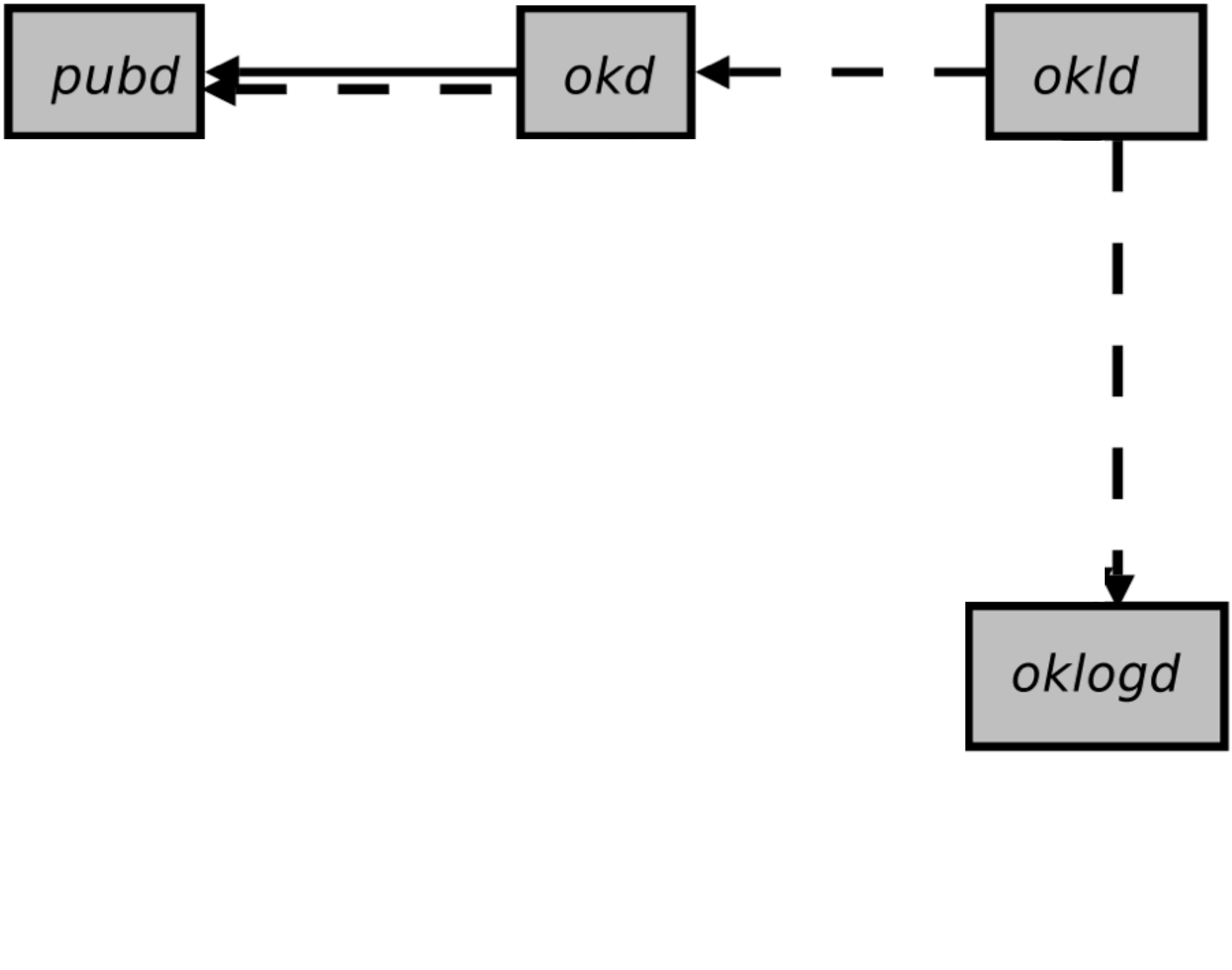
okld

Solid: RPC
Gray: HTTP
Dashed: “Parent of”
Dotted: SQL



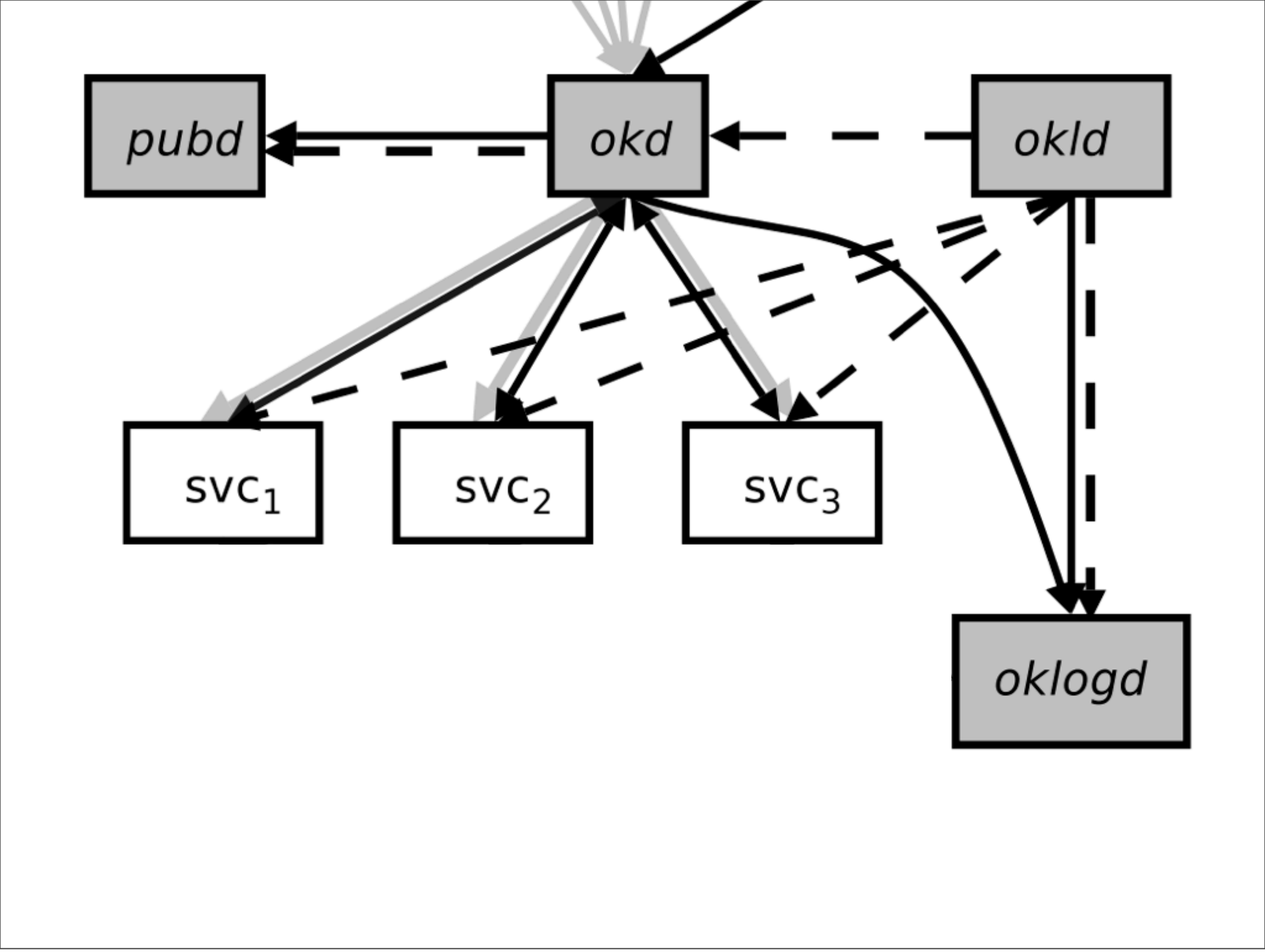
OKD: Sets up its table of regexes, basically. Received 2 sockets for LD: logging, RPC.

Solid: RPC
Gray: HTTP
Dashed: “Parent of”
Dotted: SQL



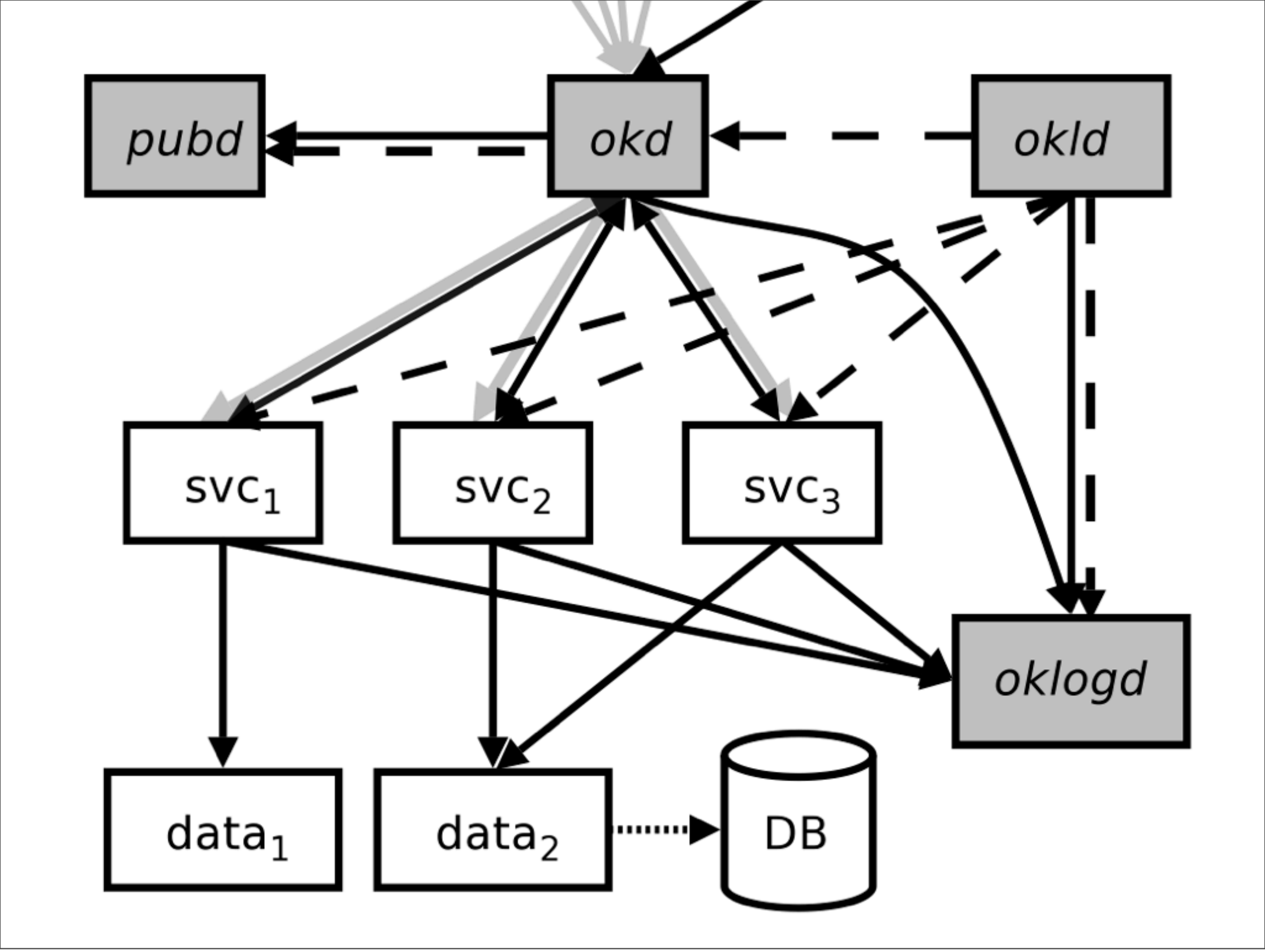
OKLD: Reads config file, opens 2 socket pairs (HTTP, RPC), forks, chroot, cd, setuid/gid/list, execs, sends server HTTP and RPC sockets to OKD.

Solid: RPC
Gray: HTTP
Dashed: “Parent of”
Dotted: SQL



OKLD: Reads config file, opens 2 socket pairs (HTTP, RPC), forks, chroot, cd, setuid/gid/list, execs, sends server HTTP and RPC sockets to OKD.

Solid: RPC
Gray: HTTP
Dashed: “Parent of”
Dotted: SQL



OKLD: Reads config file, opens 2 socket pairs (HTTP, RPC), forks, chroot, cd, setuid/gid/list, execs, sends server HTTP and RPC sockets to OKD.

Solid: RPC
Gray: HTTP
Dashed: “Parent of”
Dotted: SQL

OS Isolation

- What OS mechanisms allowed isolation?
- ...allowed access control?
  - fork, chroot, chmod, setuid, setgid
- Can we do better?
  - Capsicum: Yes, with capabilities.

Access Control

- DAC: Discretionary Access Control
  - User owns objects
  - ...can specify permissions on those (ACL, Unix)
- MAC: Mandatory Access Control
  - Objects are tagged (public, secret, top-secret)
  - Users are cleared to certain tags
  - Actions controlled by policies (secret -> public)

Capabilities

- Permissions are objects themselves
  - Basically ‘tokens’ called ‘capabilities’
- Holder of token has capabilities
- Can pass them around, give them away, etc.
- To perform action, present token

Capsicum

- An API that brings capabilities to UNIX
- Aims to enforce compartmentalization
  - Least privilege principle
- Two new primitives
  - Capabilities
  - Capability Mode

LPP = least privilege priciple

Capsicum: Capabilities

- A new type of file descriptor
  - Wraps old file descriptor
  - Create using ‘cap_new’
- Define permissions on wrapped fd
  - 60 possible masks as of writing
- Can wrap: files, dirs, procs, net, devs

Capsicum: Sandbox

- Ability to ‘cap_enter’ ‘capability mode’ ‘
  - Only access control via capabilities
- Creates a ‘clean’ environment
  - IE, removes non-capsicum fds
- Restricts system calls
- Restricts namespace

DAC vs. Capabilities

- Capabilities enable delegation
  - chroot after chrooting
- Capabilities enforce modularization
  - Send capabilities around
- Permissions on everything
  - Processes, networks, etc.

NaCl: x86 Sandbox

- Let's sandbox native, x86 code!
  - But can't it do anything?
- Conceptually, two step process:
  - 1) Verify code against validity rules
  - 2) Enforce rules, prevent side effects

NaCl Components

- Inner Sandbox
  - Validates each instruction
- Outer Sandbox
  - Intercepts system-calls
- Runtime
  - Safe API for external world access

Inner Sandbox

- Instruction level validation
- Four tenants of validation:
  - No unsafe instructions (int, lds, etc)
  - Memory access within sandbox
  - Control flow integrity (jmp to code)
  - Reliable disassembly (identify references)

Ben Bitdiddle is modifying OKWS to use Capsicum. To start each service, Ben's `okld` forks, opens the service executable binary, then calls `cap_enter()` to enter capability mode in that process, and finally executes the service binary. Each service gets file descriptors only for sockets connected to `okd`, and for TCP connections to the relevant database proxies.

3. [6 points]: Which of the following changes are safe now that the services are running under Capsicum, assuming the kernel implements Capsicum perfectly and has no other bugs?

(Circle True or False for each choice.)

A. True / False It is safe to run all services with the same UID/GID.

B. True / False It is safe to run services without `chroot`.

C. True / False It is safe to also give each service an open file descriptor for a per-service directory `/cores/servicename`.