Porting Barrelfish

Orion Hodson

Microsoft Research

Goals of this talk

1. To identify the parts that require porting
2. To help you to estimate time and effort involved
3. To provide tips and pointers to save you time

Platforms Today

X86-64

MMU Support
64-bit
Commodity Hardware

X86-32

X86-SCC

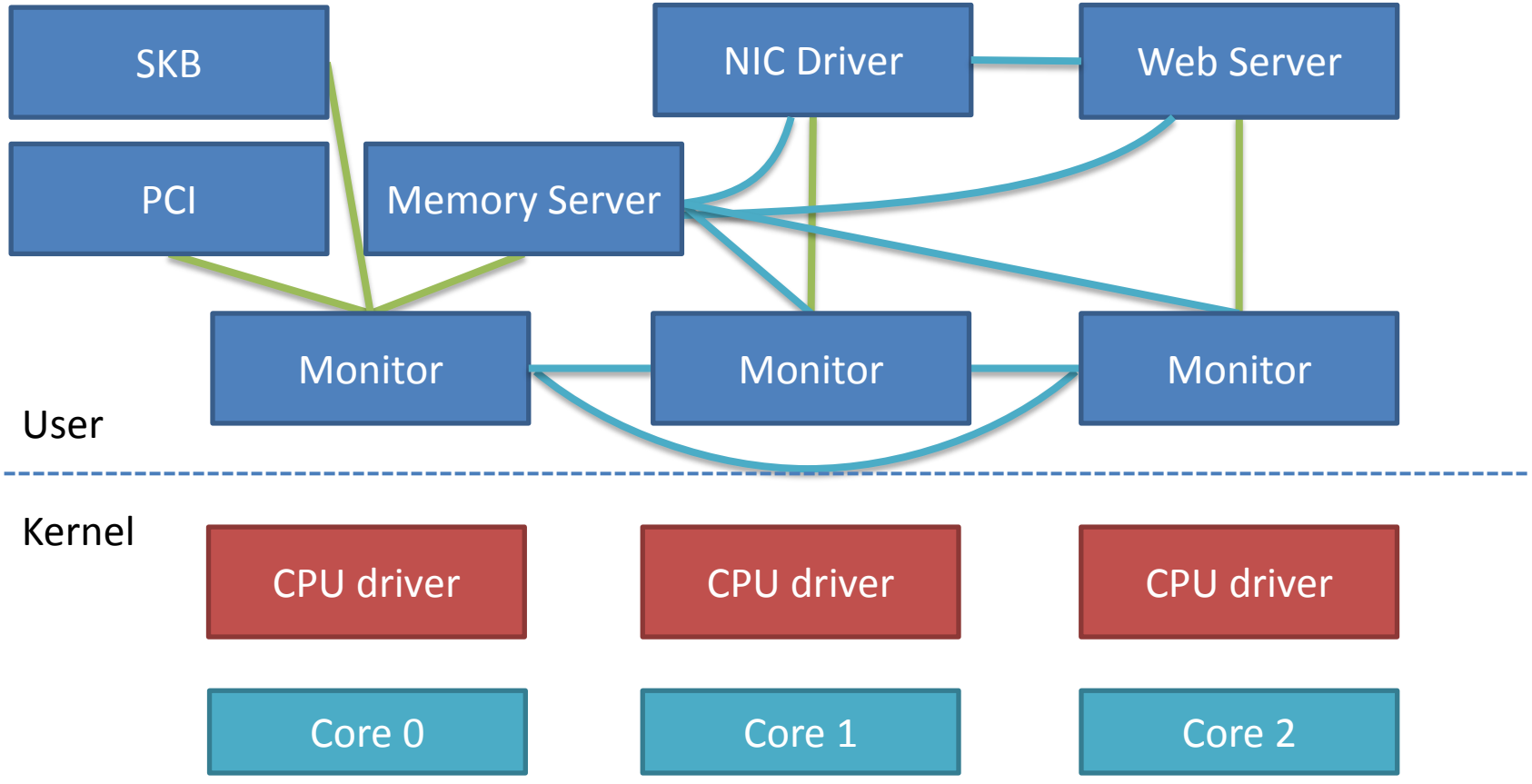
MMU Support
32-bit
Commodity Hardware

ARM v5

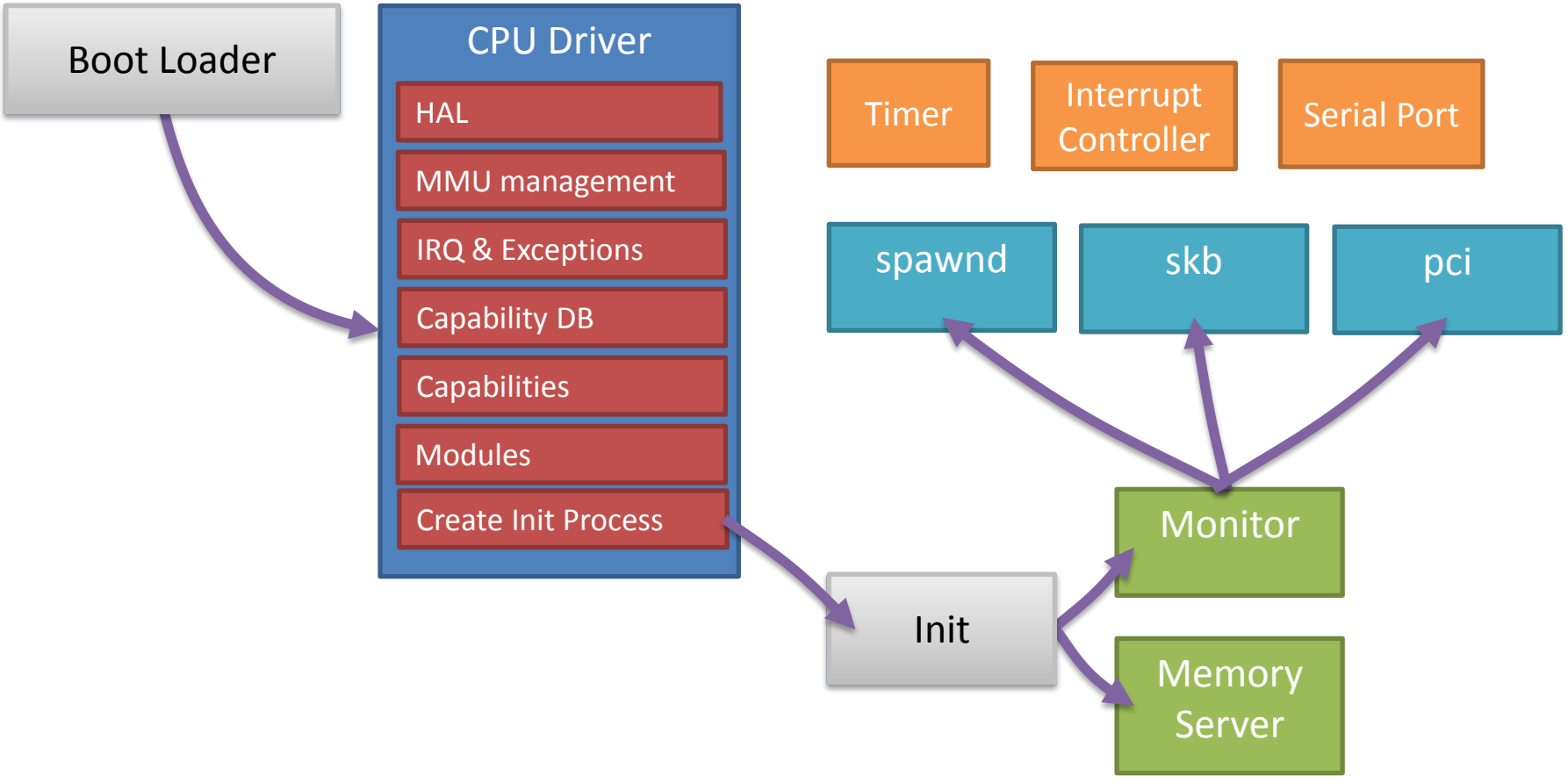
Beehive

No MMU so implements Single Address Space
32-bit RISC
FPGA and Simulated hardware

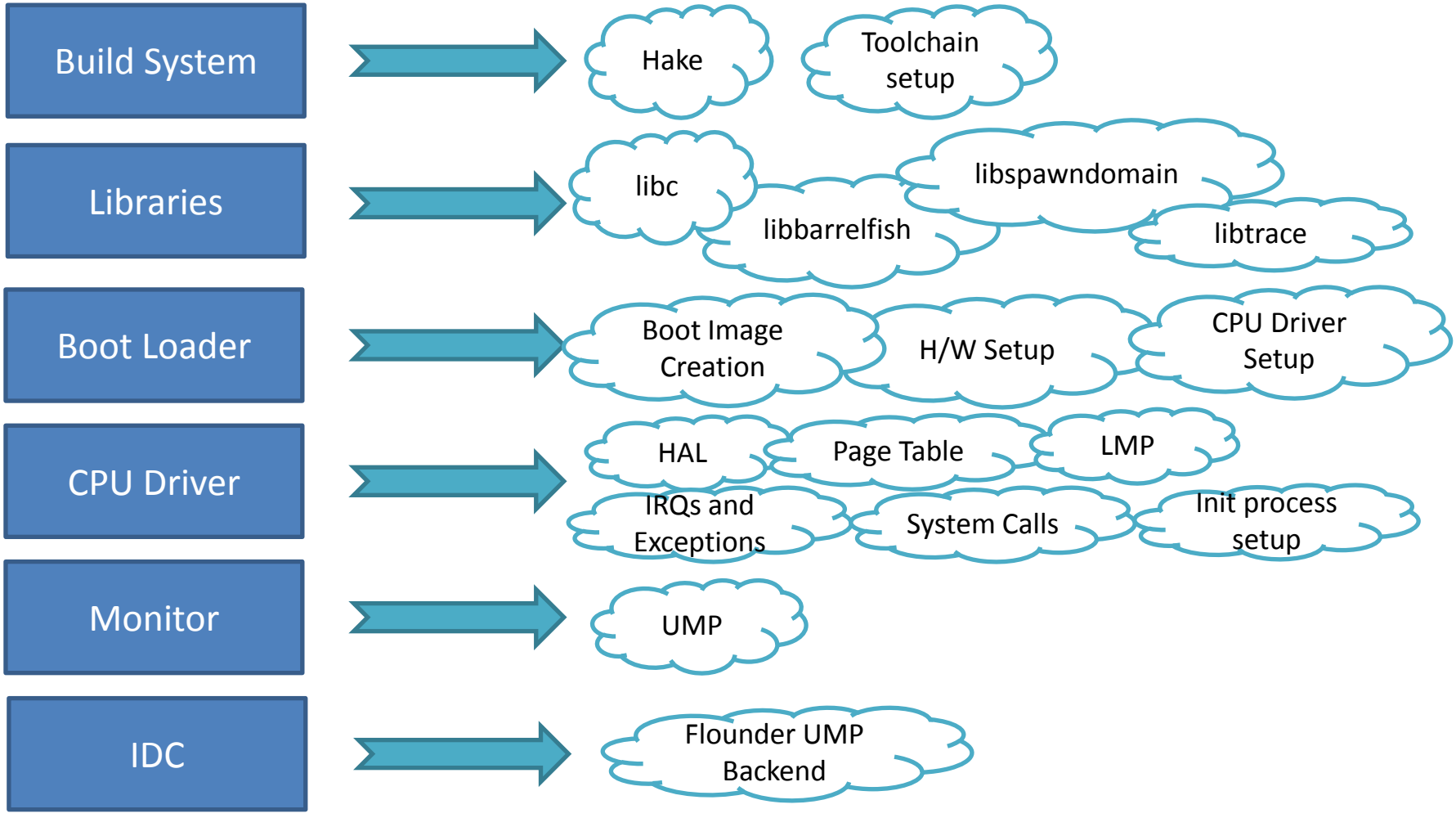
System Architecture



System Initialization



Porting Tasks



Porting Efforts

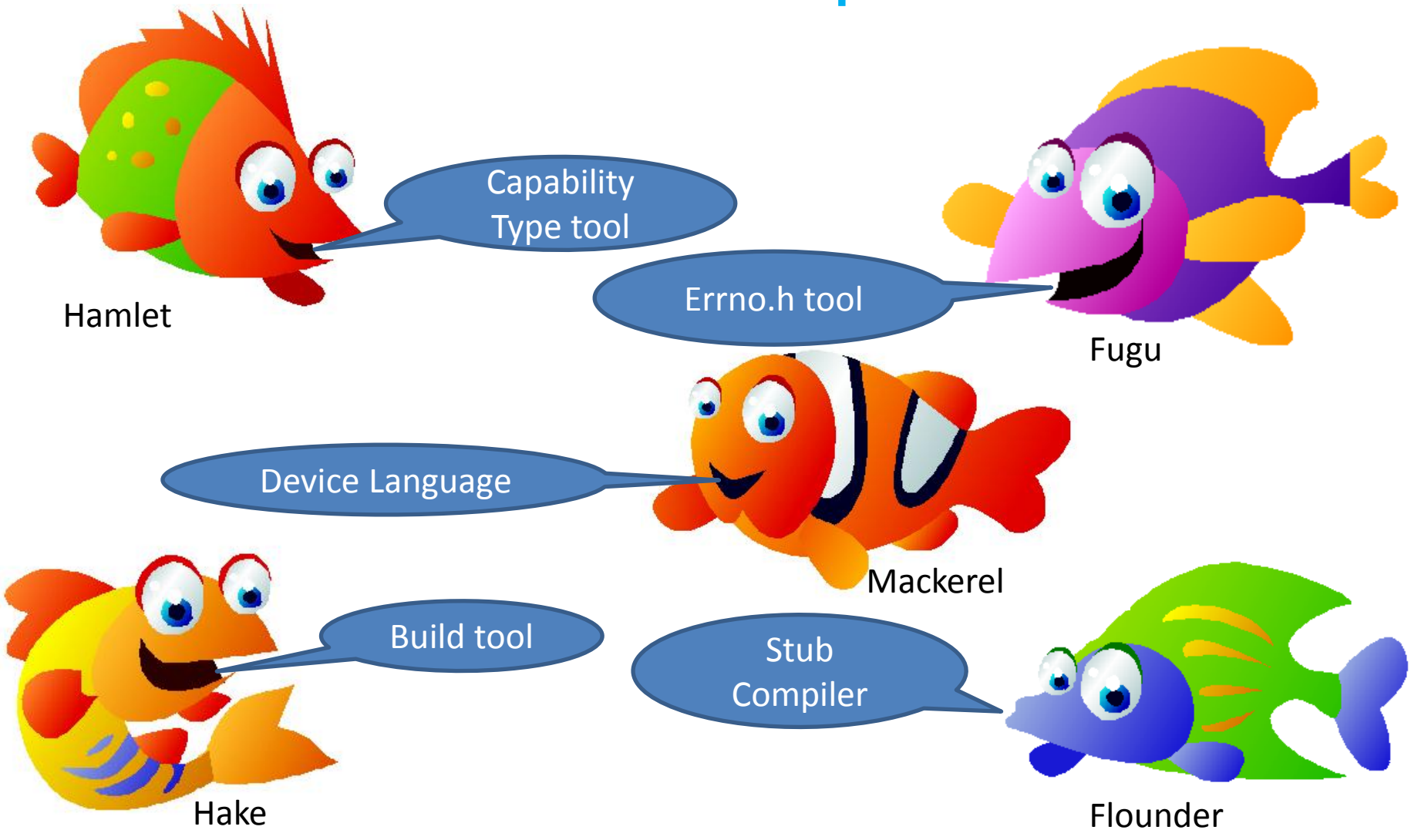
Platform	Person-months	LOC C	LOC ASM	LOC Haskell (Flounder UMP)
ARMv5	4	3597	690	TBD
Beehive	8	4397	1513	867
X86-32/SCC	2.5	7104	214	170
X86-64	-	5476	270	74
X86-Shared	-	2697	266	940

X86/SCC + X86-64 use X86-Shared code portions.
Beehive port includes time spent with experimental tools (new arch).
Lines of code counted with David A. Wheeler's SLOCCount.

Standard Build Environment

- Debian / Ubuntu Linux
- GNU toolchain (gcc, gdb, gmake, binutils)
- GHC 6.10 or 6.12.2 onwards for Barrelfish tools
  - Packages:
    - libghc6-ghc-paths-dev
    - libghc6-parsec2-dev
    - libgmp3-dev
- Mercurial for version control
- QEMU for emulation (x86,ARM)
- Cscope for source code indexing

Fish Soup


Capability
Type tool
Hamlet
Errno.h tool
Fugu
Device Language
Mackerel
Build tool
Stub
Compiler
Hake
Flounder

Hake – über Makefile generator

Hake + Hakefiles => Makefile

- `Hake` sources in:

  `$(Barrelfish)/hake`

- Platform files for different architectures:
  `ARM.hs Beehive.hs SCC.hs X86_32.hs X86_64.hs`

- `Hakefile` per project.
  - Hakefiles may contain arch specific options and files.

- `Hake` and `Hakefile` are written in Haskell.

Hakefile syntax

```
[ build application {
        Target = "pci",
        cFiles = [ "pcimain.c", "pci.c", "pci_service.c",
                   "ioapic.c", "acpi.c", "ht_config.c" ],
        flounderServers = [ "pci" ],
        mackerelDevices = [ "pci_hdr0", "pci_hdr1",
                            "lpc_ioapic", "ht_config",
                            "lpc_bridge" ],
        addIncludes  = [ "/lib/acpi/include" ],
        addLibraries = [ "mm", "pci", "acpi", "chips", "skb" ]
                     }
]
```

What gets built?

`Makefile` emitted by `Hake` includes a copy of:

```
$(Barrelfish)/hake/symbolic_targets.mk
```

in the top-level of the build directory.

This declares which binaries to build for the system. Standard list is much longer than required during early phase of port. Initially just need `cpu`, then add init_null, before working on `init`.

Also in `symbolic_targets.mk` are rules for:

```
cscope
ctags
Simulators
docs
```

`menu.lst` is boot script and must match binaries in `symbolic_targets.mk`.

Arch and Target directories

- `arch` directories contain architecture specific code.

- `target` directories contain architecture specific code that may be cross-compiled for heterogeneous systems.
  - Facilities one type of CPU manipulating system consumed by another.

Compile-Time Assertions

"Assumption is the mother of all stuff-ups"

Engineer and compiler better agree on record layouts.

```
#include <barrelfish/static_assert.h>

STATIC_ASSERT(expr, msg)
STATIC_ASSERT_SIZEOF(typename, bytes)
STATIC_ASSERT_OFFSETOF(typename, field, bytes)
```

Tools - asmoffsets

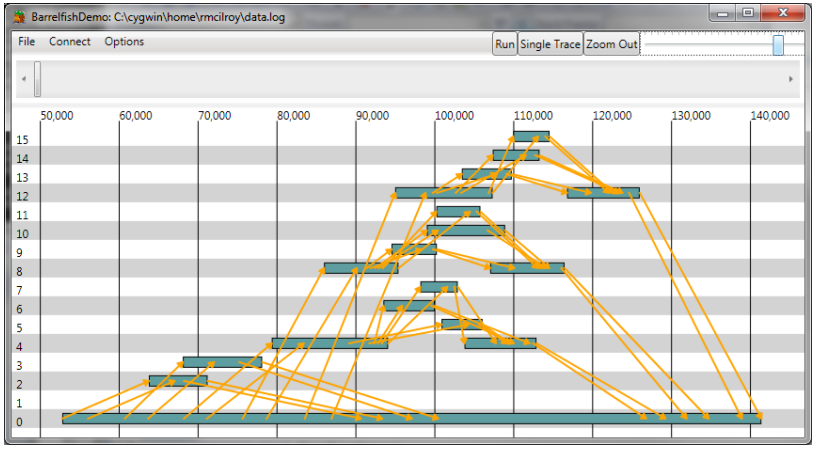
`asmoffsets` generates include files for assembler via C compiler.

- Structure sizes
- Field offsets

Tracing

Tracing library is compact.

Gives insight into behaviour.



Can be used to trace new system and existing ones.

`Aquarium` renders traces visualizations.

Dispatcher register

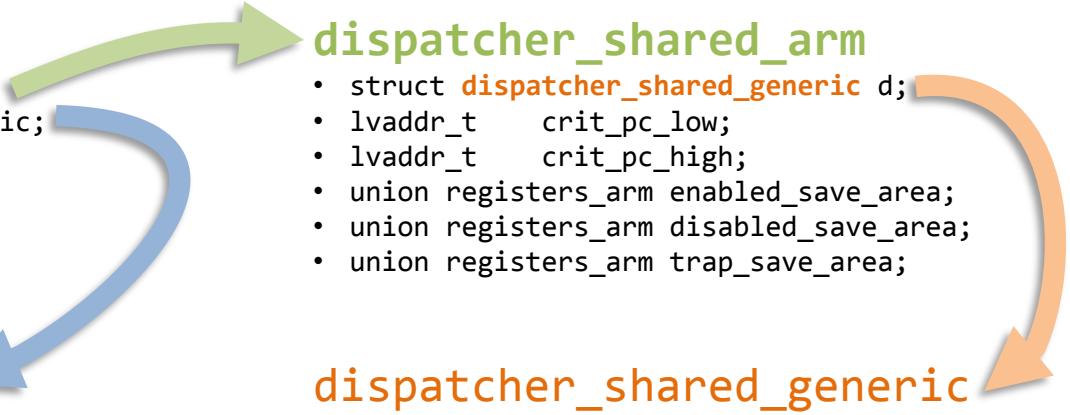
All architectures store a pointer to the current user-space dispatcher in a reserved register, e.g. FS is used on X86/X64.

Dispatcher is accessed in kernel entry points and reserving register saves work.

`curdispatcher()` returns `dispatcher_handle_t`

=> Can navigate to dispatcher structures.

Dispatcher Structures



dispatcher_arm

- struct dispatcher_shared_arm d;
- struct dispatcher_generic generic;

dispatcher_shared_arm

- struct dispatcher_shared_generic d;
- lvaddr_t crit_pc_low;
- lvaddr_t crit_pc_high;
- union registers_arm enabled_save_area;
- union registers_arm disabled_save_area;
- union registers_arm trap_save_area;

dispatcher_generic

- uintptr_t trap_stack[DISPATCHER_STACK_WORDS];
- uintptr_t stack[DISPATCHER_STACK_WORDS];
- struct thread *current;
- struct thread *runq;
- struct capref dcb_cap;
- ...
- struct trace_buffer *trace_buf;

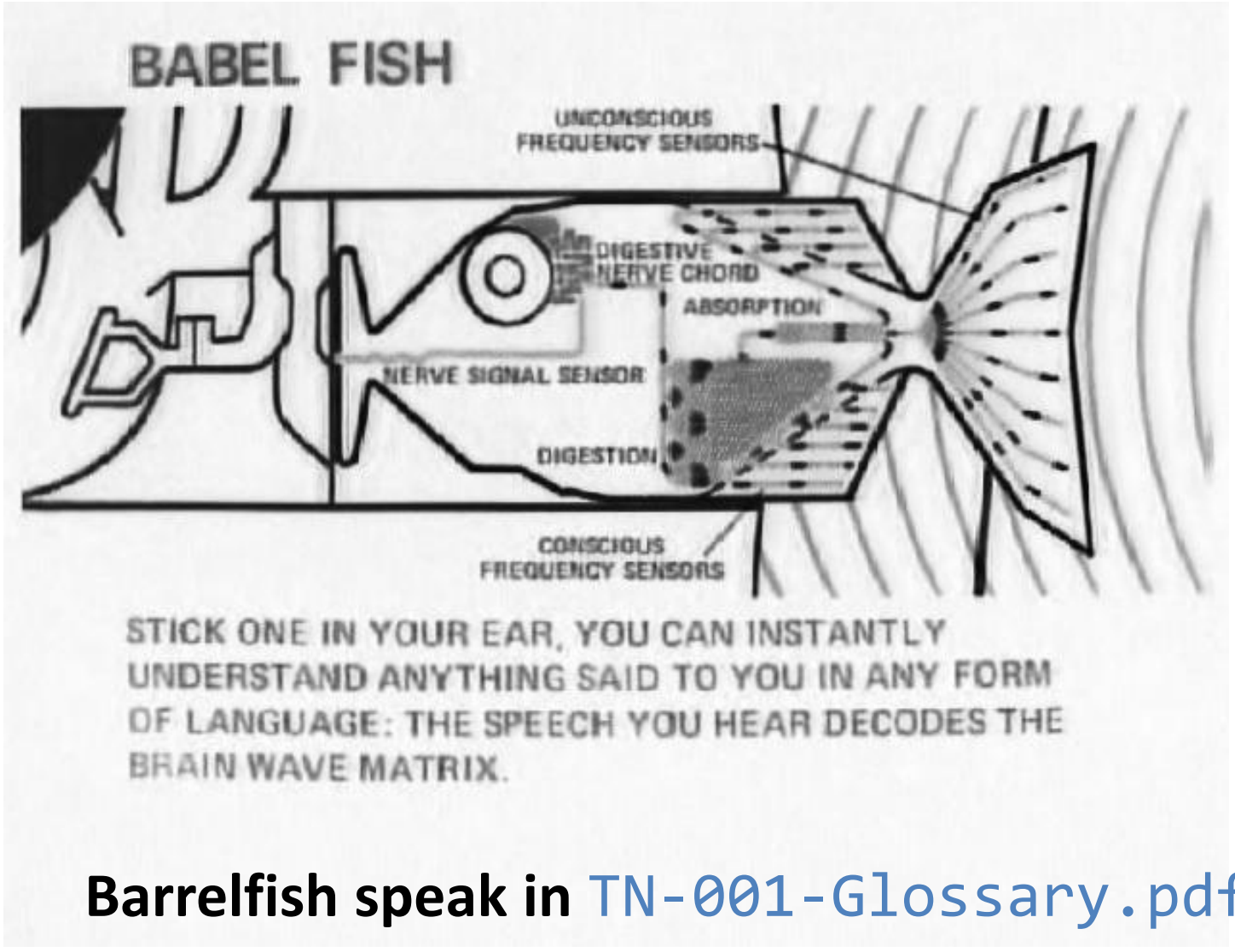
dispatcher_shared_generic

- uint32_t disabled;
- int haswork;
- lvaddr_t thread_register;
- uint32_t lmp_delivered, lmp_seen;
- lvaddr_t lmp_hint;
- lvaddr_t dispatcher_run;
- lvaddr_t dispatcher_lrpc;
- lvaddr_t dispatcher_pagefault;
- lvaddr_t dispatcher_pagefault_disabled;
- lvaddr_t dispatcher_trap;
- char name[DISP_NAME_LEN];

An Approach to Porting

- Take nearest existing system and use as template.
- Get breakpoint and logging functions (kernel/include/kernel.h) working early:
  - `breakpoint(), debug(), printk(), panic()`
- Aim for “complete” build ASAP.
  - Copy files and rename structures as needed.
  - Stub out sections that are not immediately necessary.
  - Instrument unimplemented functions with `panic(“NYI”)` so it’s obvious when the system is running outside the region you’ve reasoned about.

SUPPORT MATERIAL



Barrelfish speak in `TN-001-Glossary.pdf`