Peer-to-Peer Systems and Security

The GNUnet Architecture

Christian Grothoff

Technische Universität München

April 14, 2013

"The architects who benefit us most maybe those generous enough to lay aside their claims to genius in order to devote themselves to assembling graceful but predominantly unoriginal boxes. Architecture should have the confidence and the kindness to be a little boring." –Alain de Botton

Review: qsort

```
void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));

static int
cmpstringp(const void *p1, const void *p2) {
  return strcmp(* (char * const *) p1,
                * (char * const *) p2);
}

int main(int argc, char *argv[]) {
  qsort(&argv[1], argc - 1, sizeof(argv[1]),
        &cmpstringp);
}
```

What is GNUnet?

- GNU software package with 400k+ LOC in C
- P2P framework with focus on “security”
- Research project with over 20 related publications

Applications built using GNUnet

- Anonymous and non-anonymous file-sharing
- IPv6–IPv4 protocol translator and tunnel (P2P-based IPv6 migration)
- "The GNUnet Naming System", a censorship-resistant replacement for DNS
- SecuShare social networking application
- ...

GNUnet 0.9.x Release Status

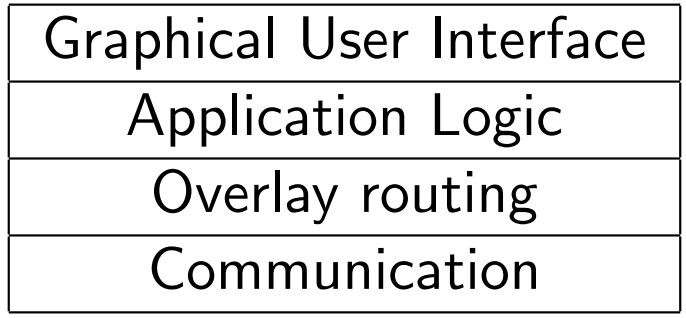
- GNUnet 0.9.5a is an alpha release
- GNUnet 0.9.5a works on GNU/Linux, OS X, W32, likely Solaris
- GNUnet 0.9.5a has known bugs (see `https://gnunet.org/bugs/`)
- GNUnet 0.9.5a lacks documentation
- GNUnet 0.9.5a has a somewhat steep learning curve

We hope to release 0.10 shortly with fewer bugs, better documentation, ...

P2P Application Needs

- Operating system abstraction layer
- Logging
- Configuration management
- Command-line parsing
- O(1)-Datastructures (heap, hash table, Bloom filter)
- Bandwidth management
- Cryptographic primitives
- Asynchronous DNS resolution

Key Layers of (most) P2P Systems



Layers in GNUnet: SecuShare

Graphical User Interface	...
Application Logic	secushare psyc psyc-db
Overlay routing	multicast mesh dht
Communication	core transport, ats udp, tcp, http

Layers in GNUnet: File-Sharing

Graphical User Interface	gnunet-fs-gtk
Application Logic	fs fs-block datastore
Overlay routing	gap mesh dht
Communication	core transport, ats udp, tcp, http

Layers in GNUnet: Protocol Translation

Graphical User Interface	gnunet-setup
Application Logic	pt exit, vpn tun
Overlay routing	regex mesh dht
Communication	core transport, ats udp, tcp, http

Layers in GNUnet: Naming System

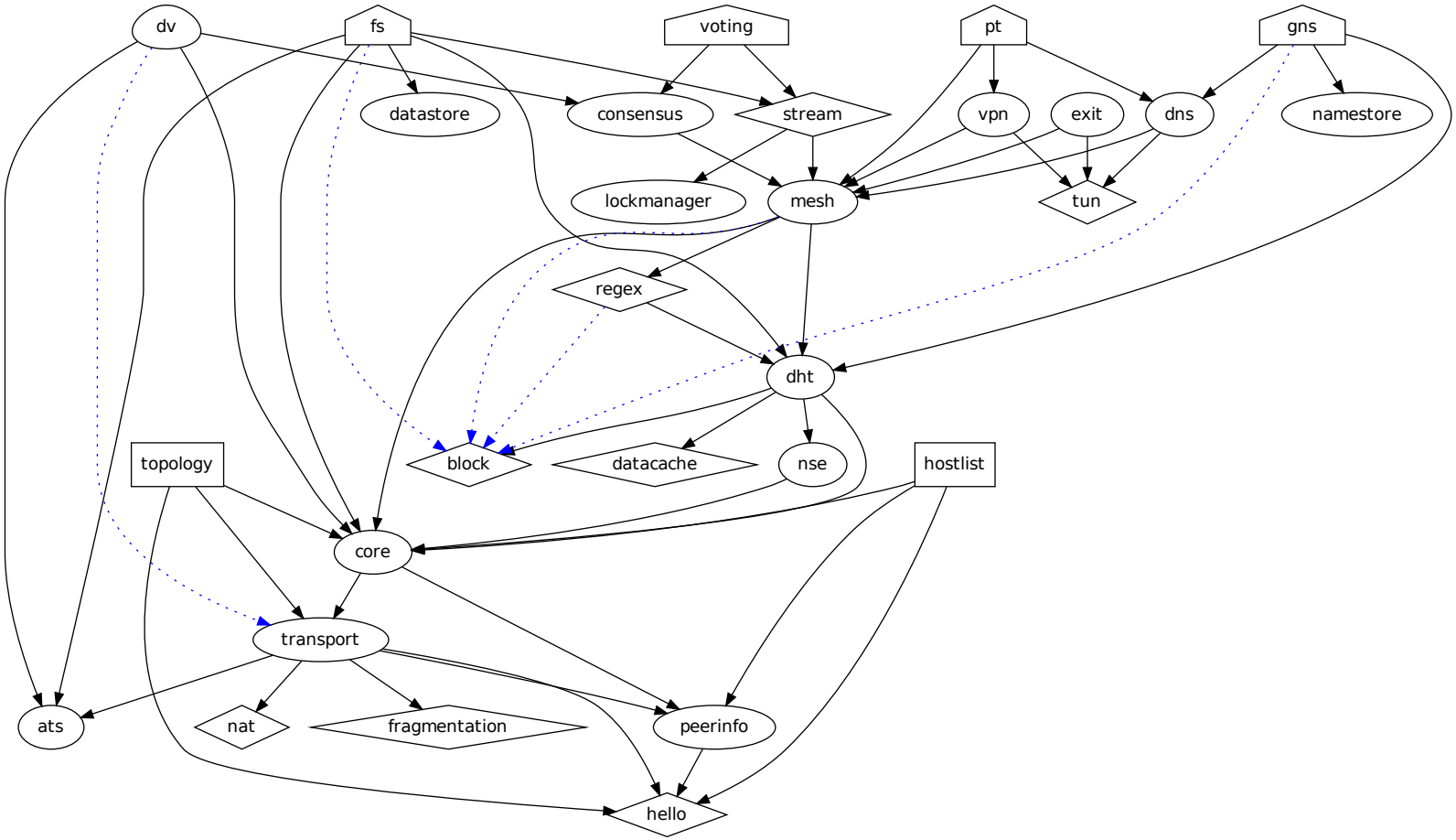
Graphical User Interface	gnunet-setup
Application Logic	gns namestore dns
Overlay routing	dht
Communication	core transport, ats udp, tcp, http

General-purpose Services

- Peer discovery (hostlist, peerinfo)
- Neighbour management (topology)
- Monitoring (statistics)
- Testing and profiling (testing, testbed)

`https://gnunet.org/gnunet-source-overview` lists all GNUnet subsystems and briefly describes their purpose.

Dependencies



GNUnet Architecture: Goals

- Security
- Extensibility
- Portability
- Performance
- Useability

GNUnet is written in C

Key concerns:

- Deadlocks, data races
- Memory corruption (stack overflow, double-free, use-after-free)
- Use of uninitialized data
- Memory leaks, socket leaks
- Arithmetic underflows and overflows, division by zero, etc.

Architecture against Insanity

Problem	Solution
Deadlocks, races	Use event loop, forbid threads
Memory corruption	Multi-process, static analysis
Uninitialized data	Wrappers around std. C functions
Memory leaks	Multi-process, dynamic analysis
Arithmetic issues	ARM, static analysis

Event-Driven Programming

- No threads
- Network communication is asynchronous
- P2P networking requires talking to many peers at once
- Clearly need to do many things at the same time!
- How can we do this without threads?

An Event Loop

Example for an event-driven application's main loop:

```
int main() {
  scheduler = create_scheduler();
  scheduler_add (scheduler, &first_task);
  while (scheduler_has_task (scheduler)) {
    task = scheduler_get_task (scheduler);
    task->run ();
  }
  destroy_Scheduler (scheduler);
}
```

The Idea

```
struct Task *scheduler_get_task () {
  wait_for = empty_event_list ();
  for (task = head; task; task = task->next)
   add_to_event_list (wait_for, task-event);
  for (task = head; task; task = task->next)
  ready = os_wait_event_ready (wait_for);
    if (ins_ready (ready, task.event))
      return task;
  return NULL;
}
```

Closer to Reality: select

```
struct Task *scheduler_get_task () {
  fd_set read_set;
  fd_set write_set;

  FD_ZERO (&read_set); FD_ZERO (&write_set);
  for (task = tasks->head; NULL != task; task = task->next) {
    if (task->wants_read) FD_ADD (&read_set, task->fd);
    if (task->wants_write) FD_ADD (&write_set, task->fd);
  }
  select (&read_set, &write_set, ...);
  for (task = tasks->head; NULL != task; task = task->next) {
    if (task->wants_read && FD_ISSET (task->fd, &read_set))
      return task;
    if (task->wants_write && FD_ISSET (task->fd, &write_set))
      return task;
  }
  return NULL; // error
}
```

Further Reading

- man 2 select
- man 2 select_tut
- man 2 poll
- man 2 epoll
- `http://www.kegel.com/c10k.html`

GNUnet API: `gnunet_scheduler_lib.h`

- Part of `libgnunetutil`
- Main event loop for GNUnet
- Each *task* is supposed to never block (disk IO is considered OK)
- Scheduler is used to schedule tasks based on IO being ready or a timeout occuring
- Each task has a unique 64-bit `GNUNET_SCHEDULER_TaskIdentifier` that can be used to *cancel* it
- The event loop is typically started using the higher-level GNUNET_PROGRAM_run or GNUNET_SERVICE_run APIs.

APIs: SHUTDOWN

The scheduler provides a somewhat tricky way to install a function that will be run on shutdown:

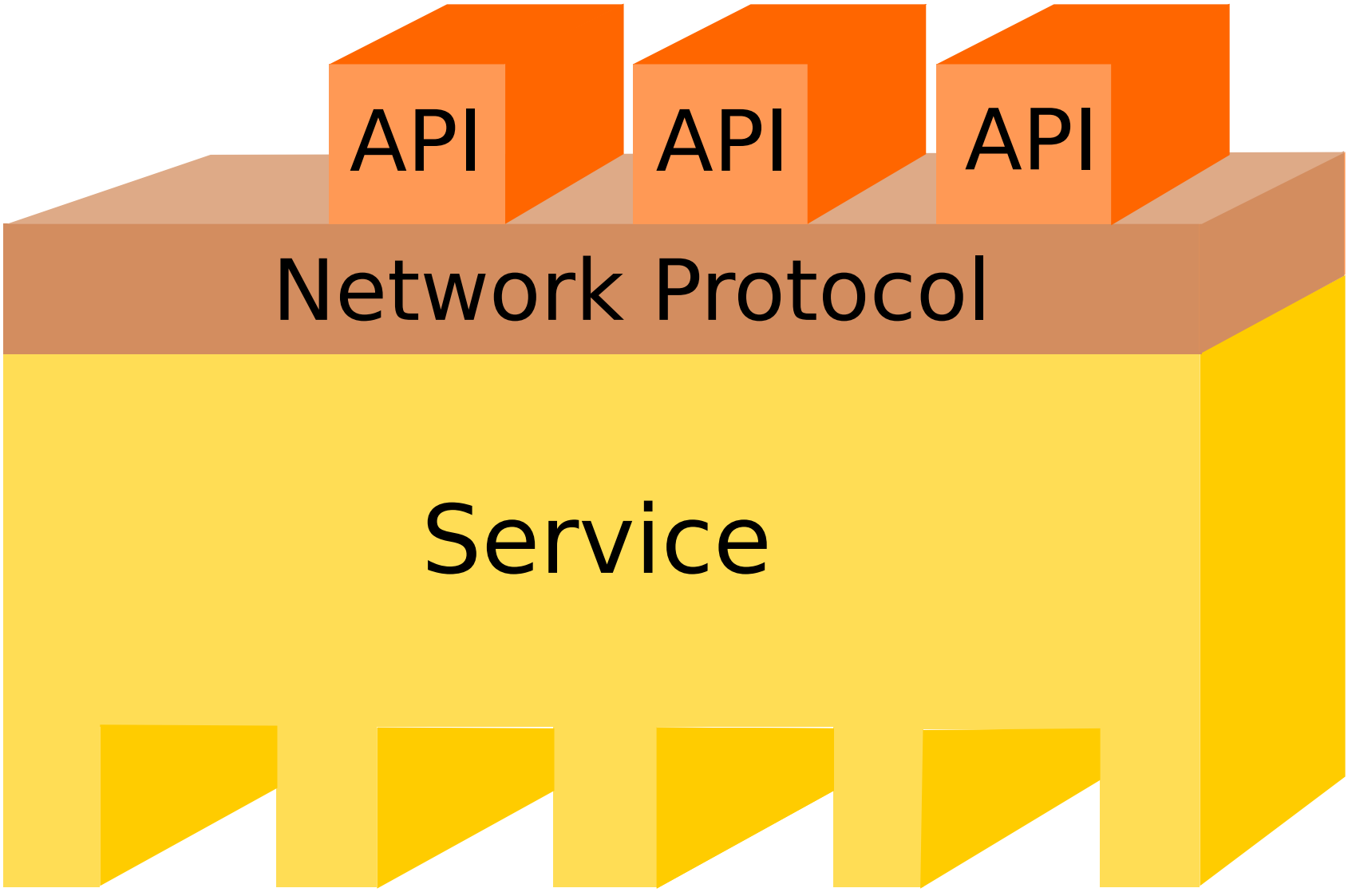
```
static void
my_shutdown (void *cls,
             const struct GNUNET_SCHEDULER_TaskContext *tc)
{
  GNUNET_assert (0 != (tc->reason & GNUNET_SCHEDULER_REASON_SHUTDOWN));
  GNUNET_CORE_disconnect (core);
}
static void
my_run (...)
{
  GNUNET_SCHEDULER_add_delayed (GNUNET_TIME_UNIT_FOREVER_REL,
                                &my_shutdown, NULL);
}
```

Reality Check

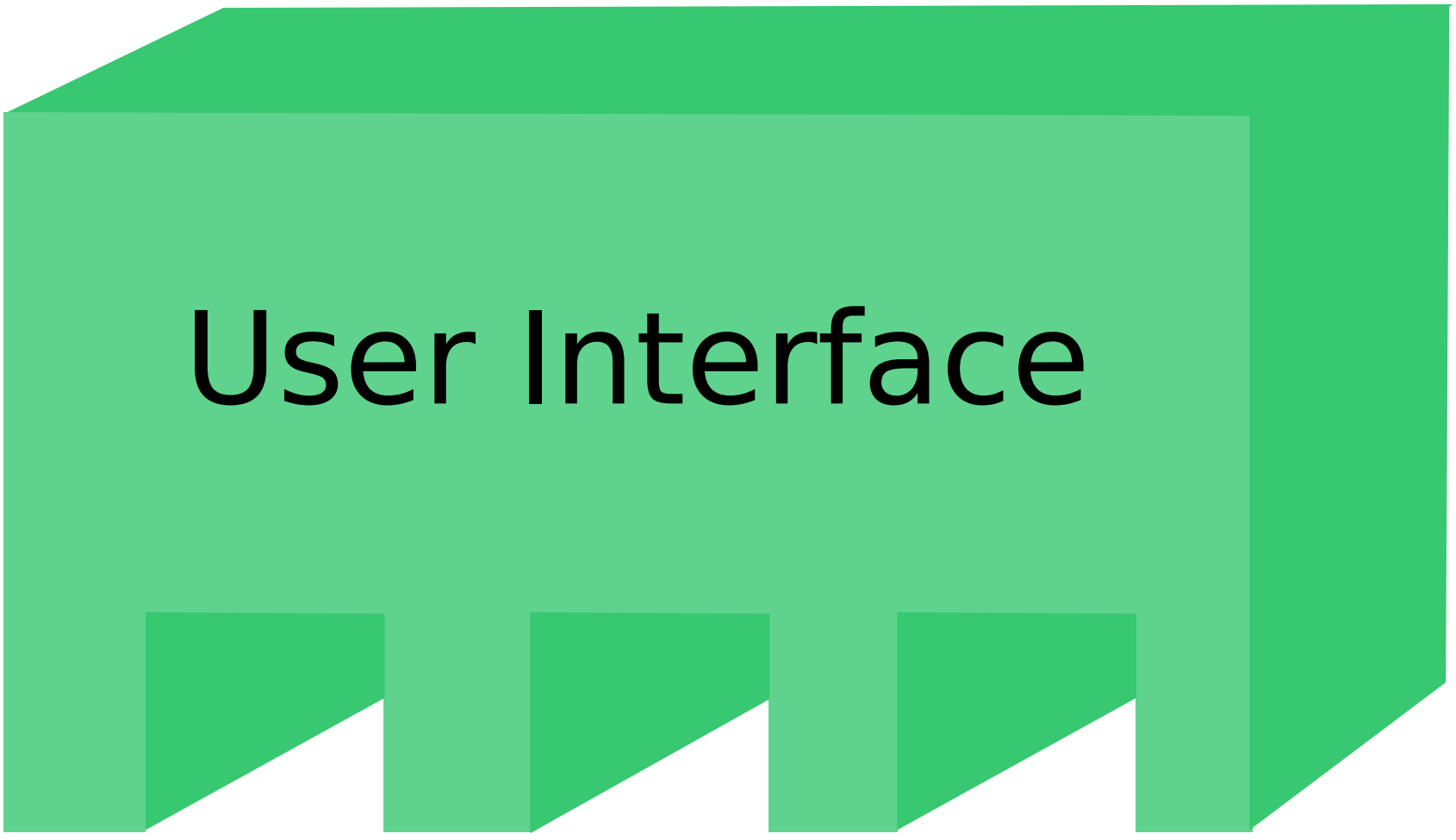
- `select` works fine for sockets (networking)
- not all APIs support event-driven programming:
  - gethostbyname
  - database APIs
  - crypto APIs
  - ...

Solution: event loops **and** processes

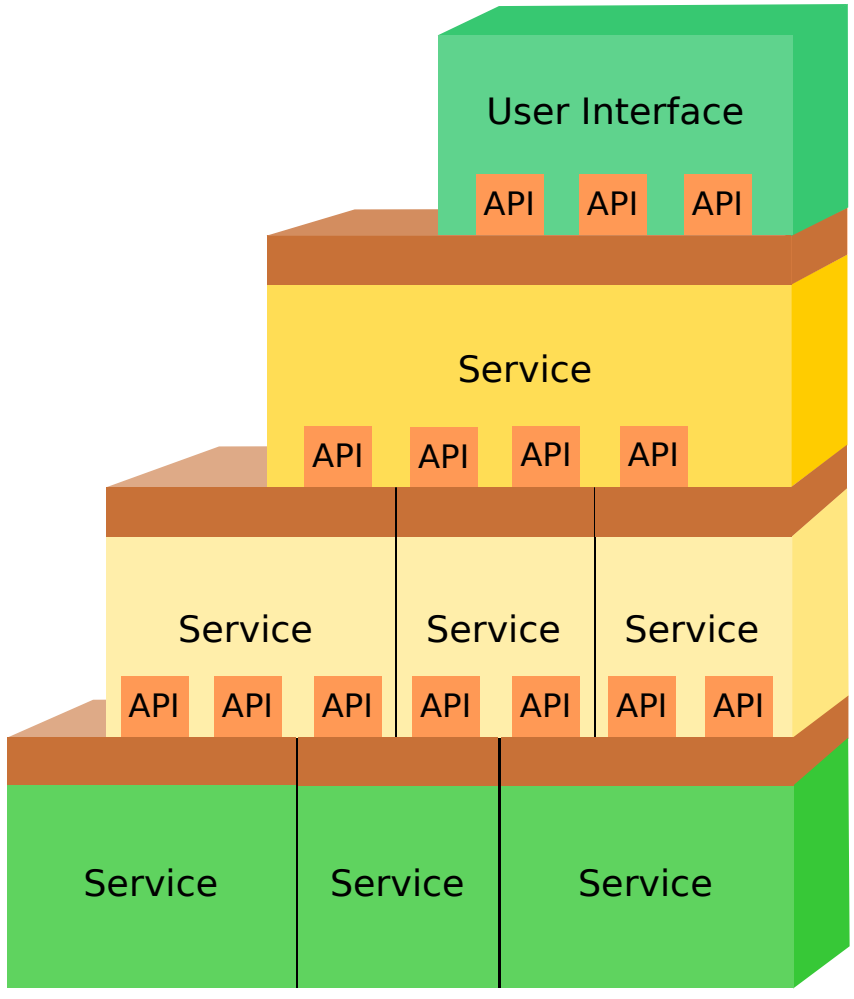
Multi-Process: A Service



Multi-Process: A Daemon



Multi-Process: A GNUnet Peer



A Typical Subsystem: statistics

- `libgnunetstatistics` library provides functions to get and set statistic values
- `gnunet_service_statistics.h` defines the public API of `libgnunetstatistics`
- `gnunet-service-statistics` binary implements server that takes requests from `libgnunetstatistics`
- `statistics.conf` specifies default configuration values for the subsystem
- `gnunet-statistics` offers a command-line interface to the service
- `gnunet-statistics.1` is a man page for the command-line tool
- `test_gnunet_statistics.py` is a test case using the command-line tool, testing also the API and the service
- `gnunet-statistics-gtk` is a GTK interface displaying statistics

Example API: gnunet_service_statistics.h

The STATISTICS service provides an easy way to track performance information:

```
struct GNUNET_STATISTICS_Handle *
GNUNET_STATISTICS_create (const char *subsystem,
                          const struct GNUNET_CONFIGURATION_Handle *cfg);
void
GNUNET_STATISTICS_set (struct GNUNET_STATISTICS_Handle *handle,
                       const char *name,
                       uint64_t value, int make_persistent);
void
GNUNET_STATISTICS_update (struct GNUNET_STATISTICS_Handle *handle,
                          const char *name,
                          int64_t delta, int make_persistent);
```

Use `gnunet-statistics` to inspect the current value of the respective statistic.

Interactions between Subsystems

- library and service communicate using TCP or UNIX Domain Sockets
- hostname, port or UNIX Domain path are specified in the configuration
- all communications use some basic meta-format
- `libgnunetutil` provides basic abstractions for the IPC

Writing a new Service

1. define header with the public API
2. define IPC protocol between library and service
3. specify default configuration for service
4. implement service library
5. implement service interaction with library
6. implement service logic
7. test, evaluate, document

A GNUnet Service is a Process

- If all subsystems are used, GNUnet would currently use 38 processes (services and daemons)
- user interfaces increase this number further
- Please start them in the correct order!

ARM

- Service processes are managed by `gnunet-service-arm`
- `gnunet-service-arm` is controlled with `gnunet-arm`
- Services are started on-demand or by-default
- Services that crash are immediately re-started
- `gnunet-arm -s` starts a peer
- `gnunet-arm -e` stops a peer

GNUnet System Overview: Help!

- `https://gnunet.org/`
  - How to build & run GNUnet
  - End-user and developer manuals, FAQ
  - Bug database
  - Doxygen source code documentation
  - Regression tests results
  - Code coverage analysis
  - Static analysis
- irc.freenode.net#gnunet

GNUnet System Overview: Dependencies

- autoconf, automake, libtool, gcc
- libgmp
- libgcrypt $\geq$ 1.5, soon $\geq$ 1.6
- gnuTLS $\geq$ 2.12.0
- libmicrohttpd $\geq$ 0.9.25
- libextractor $\geq$ 0.6.1
- libcurl $\geq$ 7.21.3
- libltdl $\geq$ 2.2
- sqlite || mysql || postgres

APIs: gnunet_util_lib.h

- Header includes many other headers
- Should be included after `platform.h`
- Provides OS independence / portability layer
- Provides higher-level IPC API (message-based)
- Provides some data structures (Bloom filter, hash map, heap, doubly-linked list)
- Provides configuration parsing
- Provides cryptographic primitives (AES-256, SHA-512, RSA, (P)RNG)
- Use: `GNUNET_malloc`, `GNUNET_free`, `GNUNET_strdup`, `GNUNET_snprintf`, `GNUNET_asprintf`, `GNUNET_log`, `GNUNET_assert`

APIs: GNUNET_assert and GNUNET_break

- GNUNET_assert aborts execution if the condition is false (0); use when internal invariants are seriously broken and continued execution is unsafe
- GNUNET_break logs an error message if the condition is false and then continues execution; use if you are certain that the error can be managed and if this has to be a programming error with the local peer
- GNUNET_break_op behaves just like GNUNET_break except that the error message blames it on other peers; use when checking that other peers are well-behaved
- GNUNET_log should be used where a specific message to the user is appropriate (not for logic bugs!); GNUNET_log_strerror and GNUNET_log_strerror_file should be used if the error message concerns a system call and errno

GNUnet Directories in Subversion

- svn/GNUnet — is GNUnet 0.8.x (do NOT use this!)
- **svn/gnunet** — is GNUnet 0.9.x
- **svn/gnunet-java** — Java bindings for GNUnet 0.9.x
- **svn/gnunet-ext** — template for writing C extensions to GNUnet
- **svn/gnunet-java-ext** — template for writing Java extensions to GNUnet
- svn/gnunet-gtk — Gtk GUIs (including gnunet-setup)
- svn/gnunet-cocoa,fuse,qt,planetlab,qt,update — experimental, defuct or legacy (ignore!)

Follow the tutorial and use **gnunet-ext**

- First figure out the build system and how to compile the existing code!
- Do change "ext" (extension) to a project-specific name everywhere
- `src/template/` in `svn/gnunet/` might also be worth a look
- Do update AUTHORS, README, etc.
- Do consider adding man pages
- Do install configuration defaults to `share/gnunet/config.d/`
- Do define your own protocol numbers (`gnunet_protocols_ext.h`)
- Feel free to add additional directories ("ext" is just a starting point)

Do you have any questions?

"The architects who benefit us most maybe those generous enough to lay aside their claims to genius in order to devote themselves to assembling graceful but predominantly unoriginal boxes. Architecture should have the confidence and the kindness to be a little boring." –Alain de Botton