Modems support in Replicant

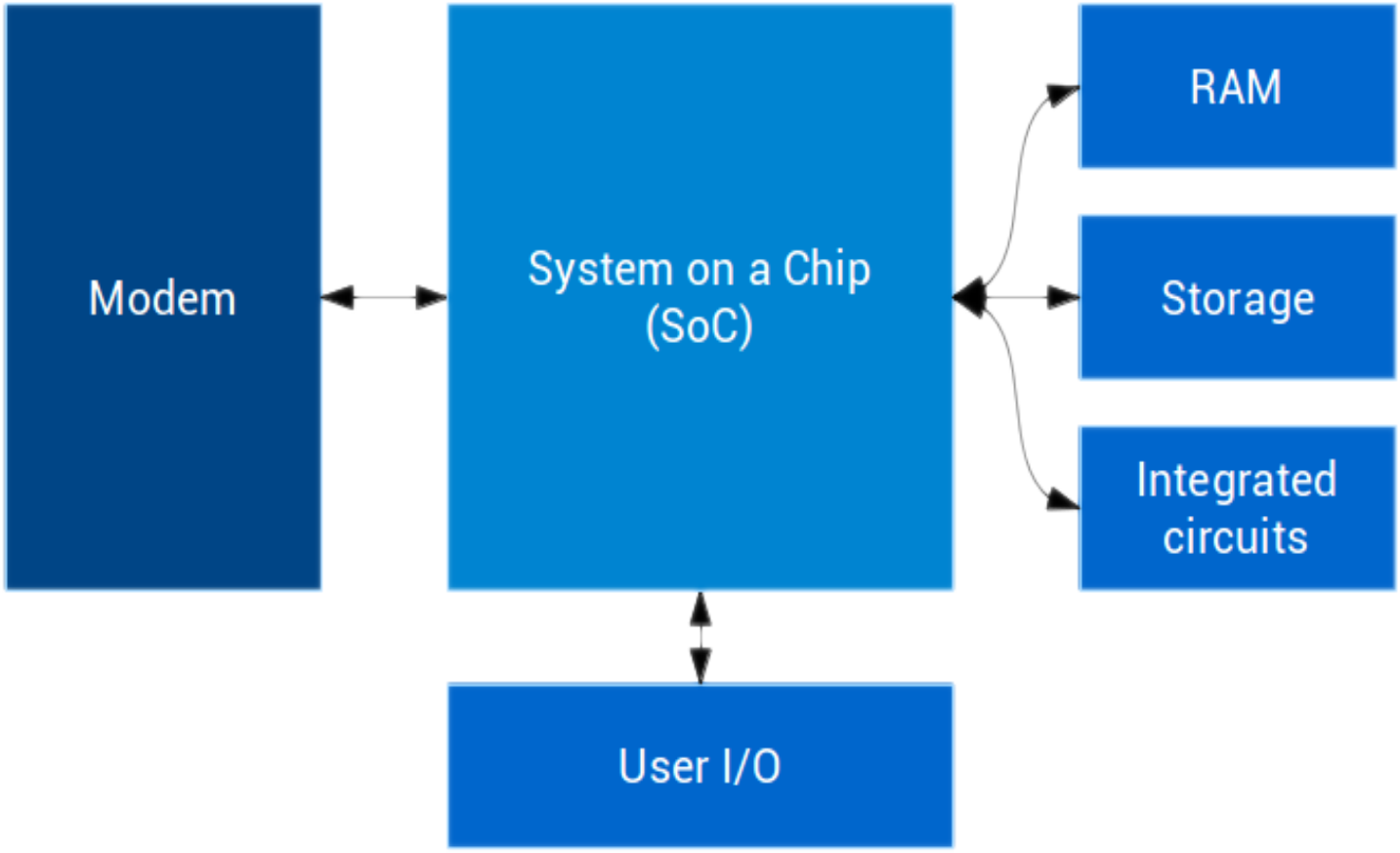
Denis 'GNUtoo' Carikli

July 28, 2019

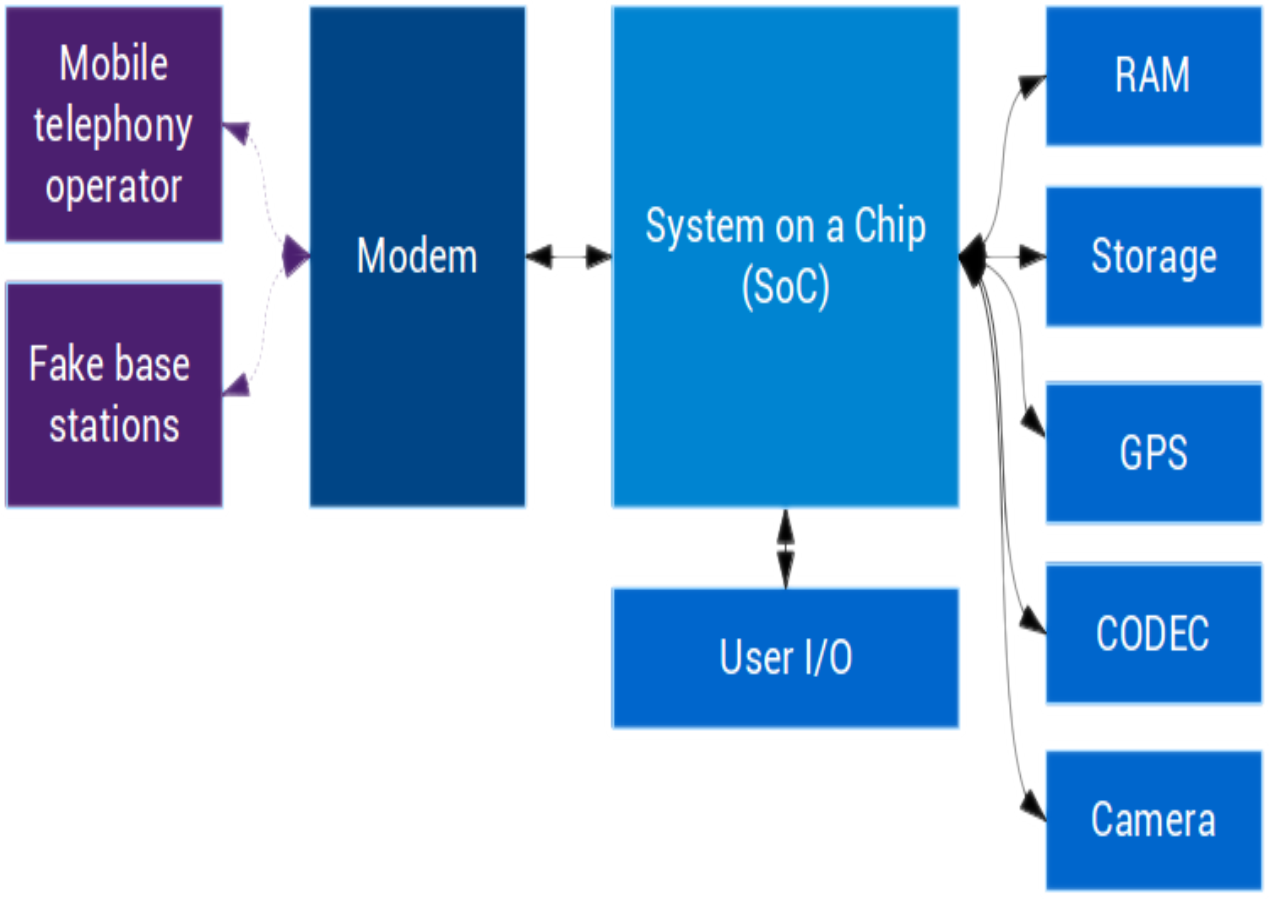
In this presentation:

- The hardware
- Android Reference implementation (More simple)
- Questions and/or Pause
- Replicant's Samsung IPC implementation (More complex)

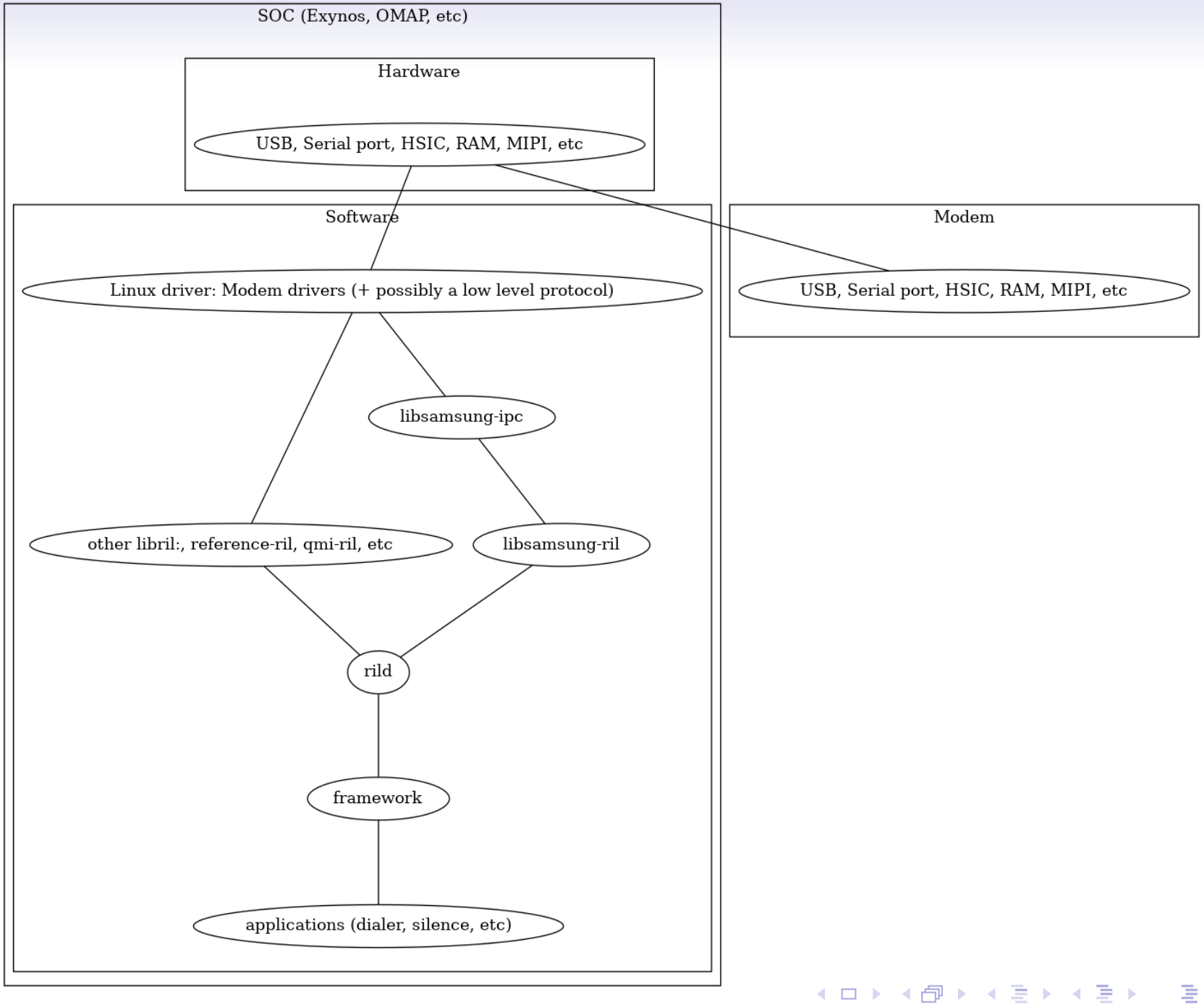
The hardware



Hardware-side overview



Good modem isolation

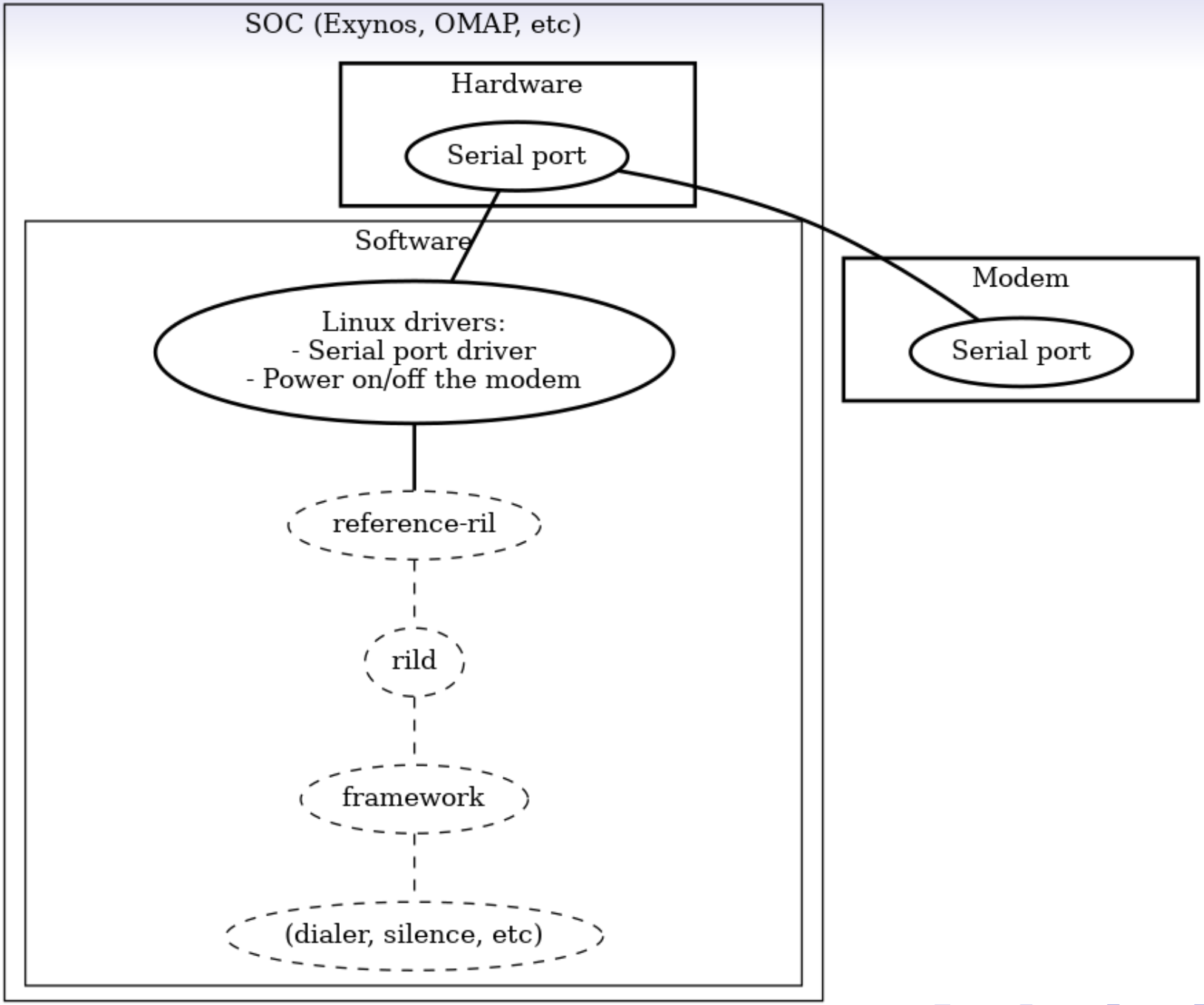

SOC (Exynos, OMAP, etc)
Hardware
USB, Serial port, HSIC, RAM, MIPI, etc
Software
Modem
Linux driver: Modem drivers (+ possibly a low level protocol)
USB, Serial port, HSIC, RAM, MIPI, etc
libsamsung-ipc
other libril:, reference-ril, qmi-ril, etc
libsamsung-ril
rild
framework
applications (dialer, silence, etc)

Why do we look at android reference implementation first?

- The hardware is more simple
- The protocol is way easier to understand
- Has good enough documentation (standard, reference implentation)
- It also explains why we got protocols like samsung-ipc
- Relevant for devices with free software bootloaders

Example: Openmoko

- Simple
- But very strongly outdated


SOC (Exynos, OMAP, etc)
Hardware
Serial port
Software
Linux drivers:
- Serial port driver
- Power on/off the modem
reference-ril
rild
framework
(dialer, silence, etc)
Modem
Serial port

AT commands

- The good parts:
  - Standard (ETSI GSM 07.07 / 3GPP TS 27.007)
  - Publically available, no registration
  - pdf versions: git://git.osmocom.org/3gpp-etsi-pdf-links.git
- The bad parts: More on that later...

Examples based on the specification (07.07)

```
# use verbose error values, report registration
> AT+CMEE=2;+CREG=1
< OK
> AT
< OK
> AT+CFUN=1
< OK
```

More complex example

```
# use verbose error values, report registration
> AT+CMEE=2;+CREG=1
< OK
> AT+CFUN=1
< +CME ERROR: SIM PIN required
> AT+CPIN="1234"
< +CME ERROR: incorrect password (user entered wrong PIN)
> AT+CPIN="4321"
< OK
# Automatic registration to an operator
# +COPS: <mode>[,<format>,<oper>]
# mode 0: automatic (<oper> field is ignored)
# format 0: long format alphanumeric <oper>
> AT+COPS=0,0
< OK
< +CREG: 1
> AT+COPS?
< +COPS: 0,0,"SFR"
> OK
```

AT commands: The bad parts:

- Syncronous
- difficult to write parsers (need to keep state)
- Slow
- Vendors extensions

More realistic example

Standard not respected

```
# 0707: +CSQ: <rssi>,<ber>
# MDM6200/6600: +CSQ: <N>
# 0707: 31 = -51dbm
# MDM6200/6600: 31 = -75dbm
> +CSQ: 31
< +CRING: VOICE/06050403002
```

Vendor specific commands

```
# Enable noise cancelation on the OpenMoko
> AT%N0105
< OK
```

Example of issues

- What if the answer doesn't come back?
- What if there is some noise on the serial port?
- What if I need to run another commands while waiting for the answer of the previous one?
- More modern modems with AT protocol (Example: GTA04):
  - Modem connected over USB
  - Serveral "virtual serial" interfaces

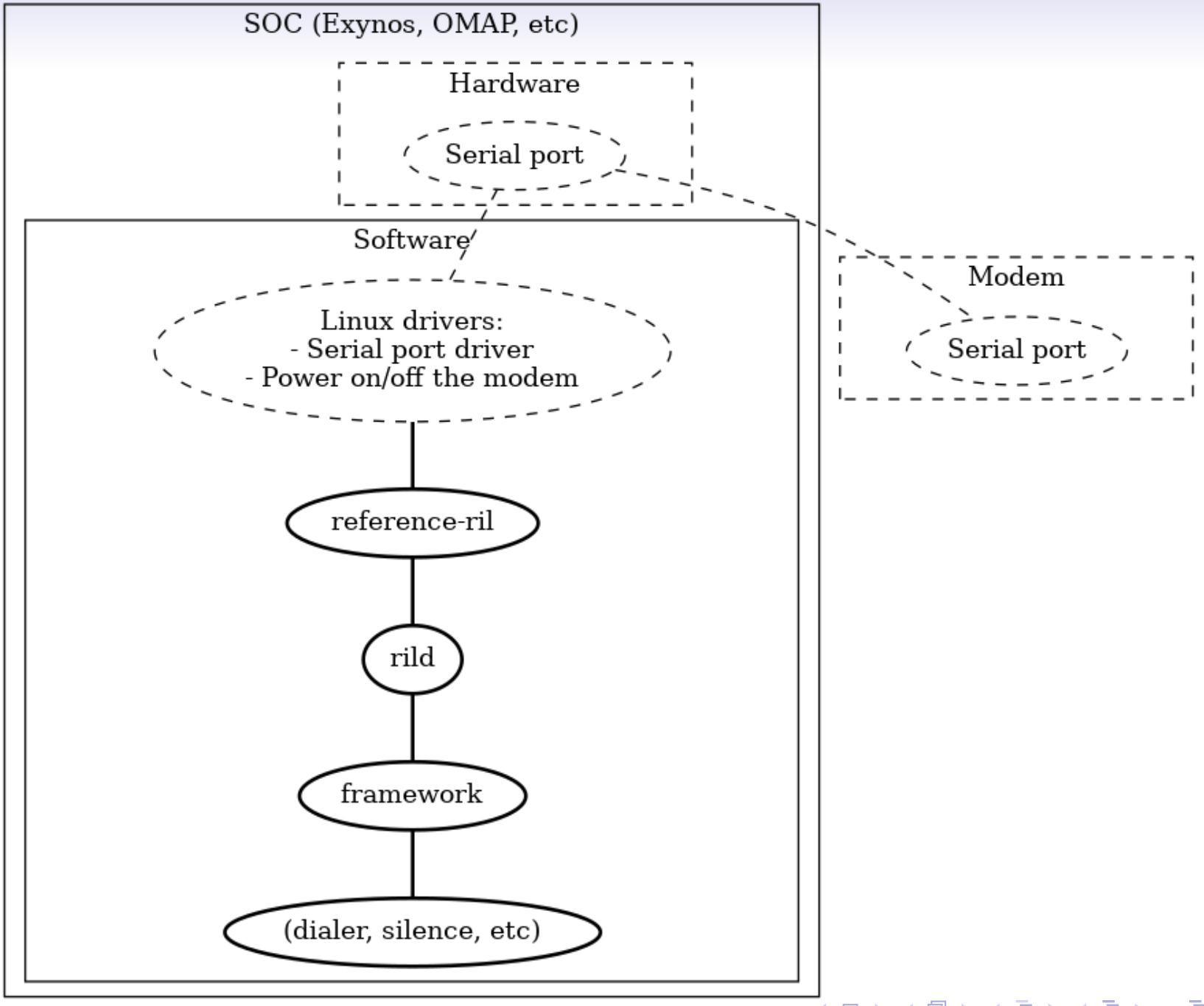
The bad parts:

- $\rightarrow$This lead modem and device vendors to make their own protocol.
- $\rightarrow$Very similar issues with GPS (NMEA not meant for GPS $\rightarrow$ custom protocols by modem chip vendors (UBlox, SIRF, etc)).

AT commands are still in use:

- GTA04
- Optimus black
- Usually (also) available on modems available at low quantity orders.
- $\rightarrow$Often in use on devices made for the free software community .
- Sometimes vendor documentation is even publically available for specific modems.

Handling AT command set in Android


SOC (Exynos, OMAP, etc)
Hardware
Serial port
Software
Linux drivers:
- Serial port driver
- Power on/off the modem
reference-ril
rild
framework
(dialer, silence, etc)
Modem
Serial port

rild

finding libril implementation: hardware/ril/rild/rild.c

```
int main(int argc, char **argv) {
    // vendor ril lib path either passed in as
    // -l parameter, or read from rild.libpath
    // property
    const char *rilLibPath = NULL;
    // ril arguments either passed in
    // as -- parameter, or read from rild.libargs
    // property
    ...
}
```

finding libril implementation

```
$ cd device/samsung/i9300
$ git grep rild
system.prop:rild.libpath=/system/lib/hw/libsamsung-ril.so
system.prop:rild.libargs=-d /dev/ttyS0
```

In the code: hardware/ril/rild/rild.c

```
int main(int argc, char **argv) {
    ...
    const RIL_RadioFunctions *(*rilInit)(
            const struct RIL_Env *, int, char **);
    ...
    dlHandle = dlopen(rilLibPath, RTLD_NOW);
    ...
    RIL_startEventLoop();
    ...
    rilInit = (const RIL_RadioFunctions *(*)(
                const struct RIL_Env *, int, char **))
                dlsym(dlHandle, "RIL_Init");
    funcs = rilInit(&s_rilEnv, argc, rilArgv);
    ...
    RIL_register(funcs);
    ...
}
```

reference-ril

- Implements libril, like libsamsung-ril
- Same git repository than rild
- AT commands, very basic (only one channel)
- Beware of CaMeL Case Code and _ mix (RIL_Init, not RIL_init)

```
static const RIL_RadioFunctions s_callbacks = {
    RIL_VERSION,
    onRequest,
    currentState,
    onSupports,
    onCancel,
    getVersion
};
...

const RIL_RadioFunctions *RIL_Init(
                            const struct RIL_Env *env,
                            int argc, char **argv) {
    ...
    return &s_callbacks;
}
```

Calling

```
static void onRequest (int request, void *data,
size_t datalen, RIL_Token t) {
  ...
  switch (request) {
    ...
    case RIL_REQUEST_DIAL:
    requestDial(data, datalen, t);
    break;
    case ...
  }
```

requestDial

```
static void requestDial(void *data,
size_t datalen __unused, RIL_Token t) {
    ...
    ret = at_send_command(cmd, NULL);

    free(cmd);
    ...
    RIL_onRequestComplete(t, RIL_E_SUCCESS,
    NULL, 0);
}
```

Incomming call

onUnsolicited

```
static void onUnsolicited (const char *s,
                           const char *sms_pdu) {
    ...
    if (strStartsWith(s,"+CRING:")
    || strStartsWith(s,"RING")
    || strStartsWith(s,"NO CARRIER")
    || strStartsWith(s,"+CCWA") ) {
        RIL_onUnsolicitedResponse (
        RIL_UNSOL_RESPONSE_CALL_STATE_CHANGED,
        NULL, 0);
        ...
    }
    ...
}
```

RIL_Init

```
const RIL_RadioFunctions *RIL_Init(
const struct RIL_Env *env,
int argc, char **argv) {
    ...
    ret = pthread_create(&s_tid_mainloop, &attr,
    mainLoop, NULL);
    ...
}
```

MainLoop

```
static void * mainLoop(void *param __unused) {
    ...
    ret = at_open(fd, onUnsolicited);
    ...
}
```

at_open

```
static ATUnsolHandler s_unsolHandler;
...

int at_open(int fd, ATUnsolHandler h)
{
    ...
    s_unsolHandler = h;
    ...
    ret = pthread_create(&s_tid_reader, &attr,
                                        readerLoop, &attr);
    ...
}
```

readerLoop

```
static void *readerLoop(void *arg __unused)
{
  for (;;) {
    ...
    processLine(line);
    ...
  }
  ...
}
```

processLine

```
    static void processLine(const char *line) {
    pthread_mutex_lock(&s_commandmutex);
    if (sp_response == NULL) {
        /* no command pending */
        handleUnsolicited(line);
    } ...
}
```

```
    static void handleUnsolicited(const char *line)
    {
        if (s_unsolHandler != NULL) {
            s_unsolHandler(line, NULL);
        }
    }
```

Other protocols

Other protocols: status

- "samsung-ipc"
  - Implemented in libsamsung-ipc
  - Usable on Android and GNU/Linux
  - Incomplete
  - No wireshark dissectror
  - No upstream Linux driver yet
- QMI
  - Implemented in libqmi, other?
  - Usable with upstream Linux and in GNU/Linux
- ISI (Nokia: N900, N9, etc)
  - Implemented in Ofono, Freesmartphone.org, other?
  - Wireshark dissector
  - Upstream Linux drivers
- "Palm Pre"
  - Implemented in msmcomm
  - Was usable in GNU/Linux through Freesmartphone.org
  - Code lost? Michael Lauer should push it again soon.

Other protocols: characteristics

- Like a network protocol
  - sequence number to match query and response
  - Asyncronous
- Free software implementation not always available
- Or incomplete (samsung-ipc)

Use a supported protocol

- Example with the Palm pre
  - Had an unknown protocol that was used by the nonfree default implementation
  - Also had AT commands
  - People implemented the AT commands
  - No ring indication if my memory is correct
  - At the end they implemented the unknown vendor protocol in msmcommd

Questions?

- Next part is about Samsung IPC
- Increasing level of complexity
- Increasing level of complexity
- $\rightarrow$ Questions on the first part before continuing.

Licenses:

- Attribution-ShareAlike 3.0 Unported for the two images that comes from the Freedom Privacy and Security page.
- `https://creativecommons.org/licenses/by-sa/4.0/` for the rest.