Running a basic Osmocom GSM network

Harald Welte <laforge@gnumonks.org>

What this talk is about

- Implementing GSM/GPRS network elements as FOSS
- Applied Protocol Archaeology
- Doing all of that on top of Linux (in userspace)

Running your own Internet-style network

- use off-the-shelf hardware (x86, Ethernet card)
- use any random Linux distribution
- configure Linux kernel TCP/IP network stack
  - enjoy fancy features like netfilter/iproute2/tc
- use apache/lighttpd/nginx on the server
- use Firefox/chromium/konqueor/lynx on the client
- do whatever modification/optimization on any part of the stack

Running your own GSM network

Until 2009 the situation looked like this:

- go to Ericsson/Huawei/ZTE/Nokia/Alcatel/...
- spend lots of time convincing them that you're an eligible customer
- spend a six-digit figure for even the most basic full network
- end up with black boxes you can neither study nor improve
  - WTF?
  - I've grown up with FOSS and the Internet. I know a better world.

Why no cellular FOSS?

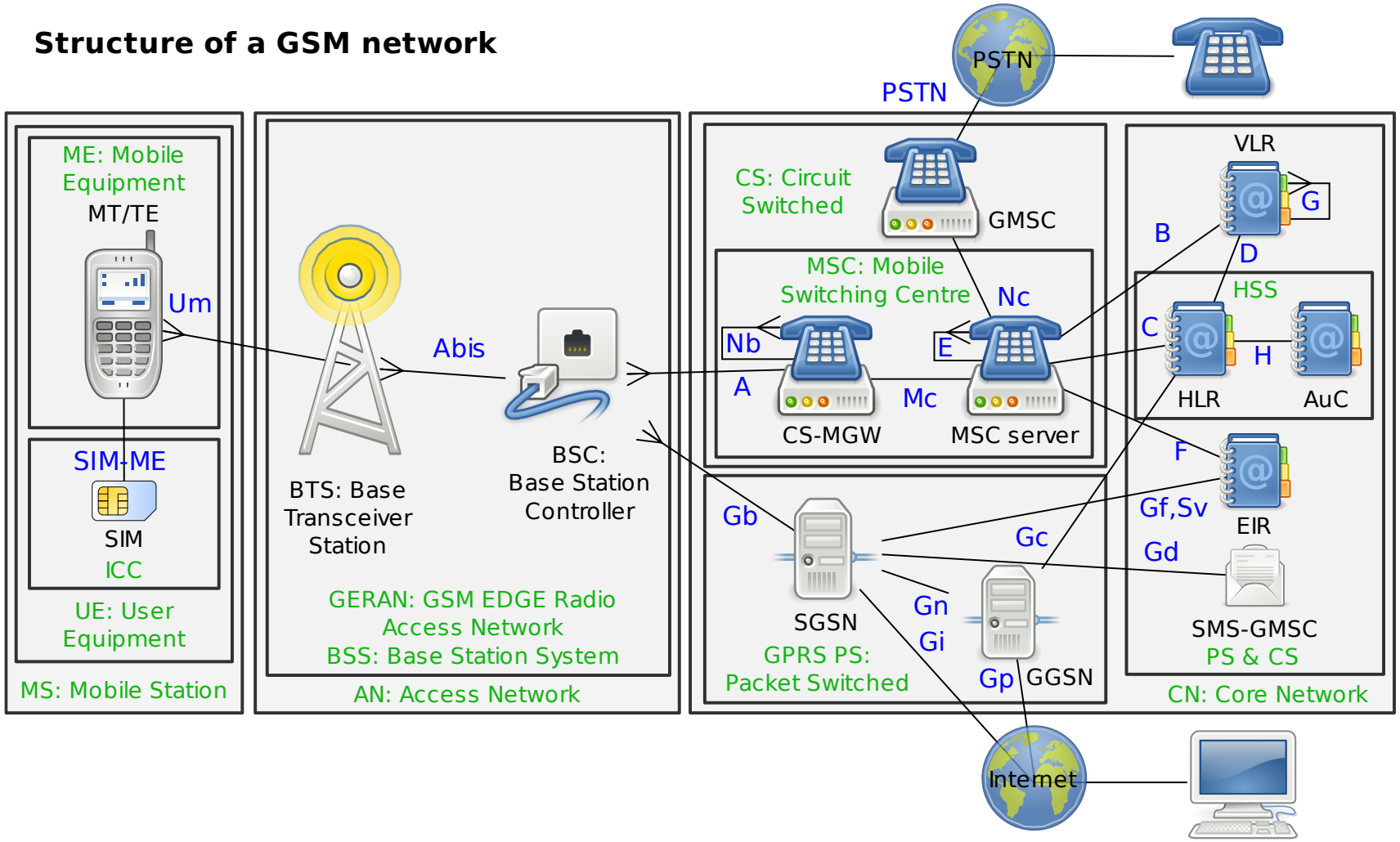
- both cellular (2G/3G/4G) and TCP/IP/HTTP protocol specs are publicly available for decades. Can you believe it?
- Internet protocol stacks have lots of FOSS implementations
- cellular protocol stacks have no FOSS implementations for the first almost 20 years of their existence?
- it's the classic conflict
  - classic circuit-switched telco vs. the BBS community
  - ITU-T/OSI/ISO vs. Arpanet and TCP/IP

Enter Osmocom

In 2008, some people (most present in this room) started to write FOSS for GSM

- to boldly go where no FOSS hacker has gone before
  - where protocol stacks are deep
  - and acronyms are plentiful
  - we went from `bs11-abis` to `bsc_hack` to *OpenBSC*
  - many other related projects were created
  - finally leading to the *Osmocom* umbrella project

Classic GSM network architecture



GSM Acronyms, Radio Access Network

MS

Mobile Station (your phone)

BTS

Base Transceiver Station, consists of 1..n TRX

TRX

Transceiver for one radio channel, serves 8 TS

TS

Timeslots in the GSM radio interface; each runs a specific combination of logical channels

BSC

Base Station Controller

GSM Acronyms, Core Network

MSC

Mobile Switching Center; Terminates MM + CC Sub-layers

HLR

Home Location Register; Subscriber Database

SMSC

SMS Service Center

GSM Acronyms, Layer 2 + 3

LAPDm

Link Access Protocol, D-Channel. Like LAPD in ISDN

RR

Radio Resource (establish/release dedicated channels)

MM

Mobility Management (registration, location, authentication)

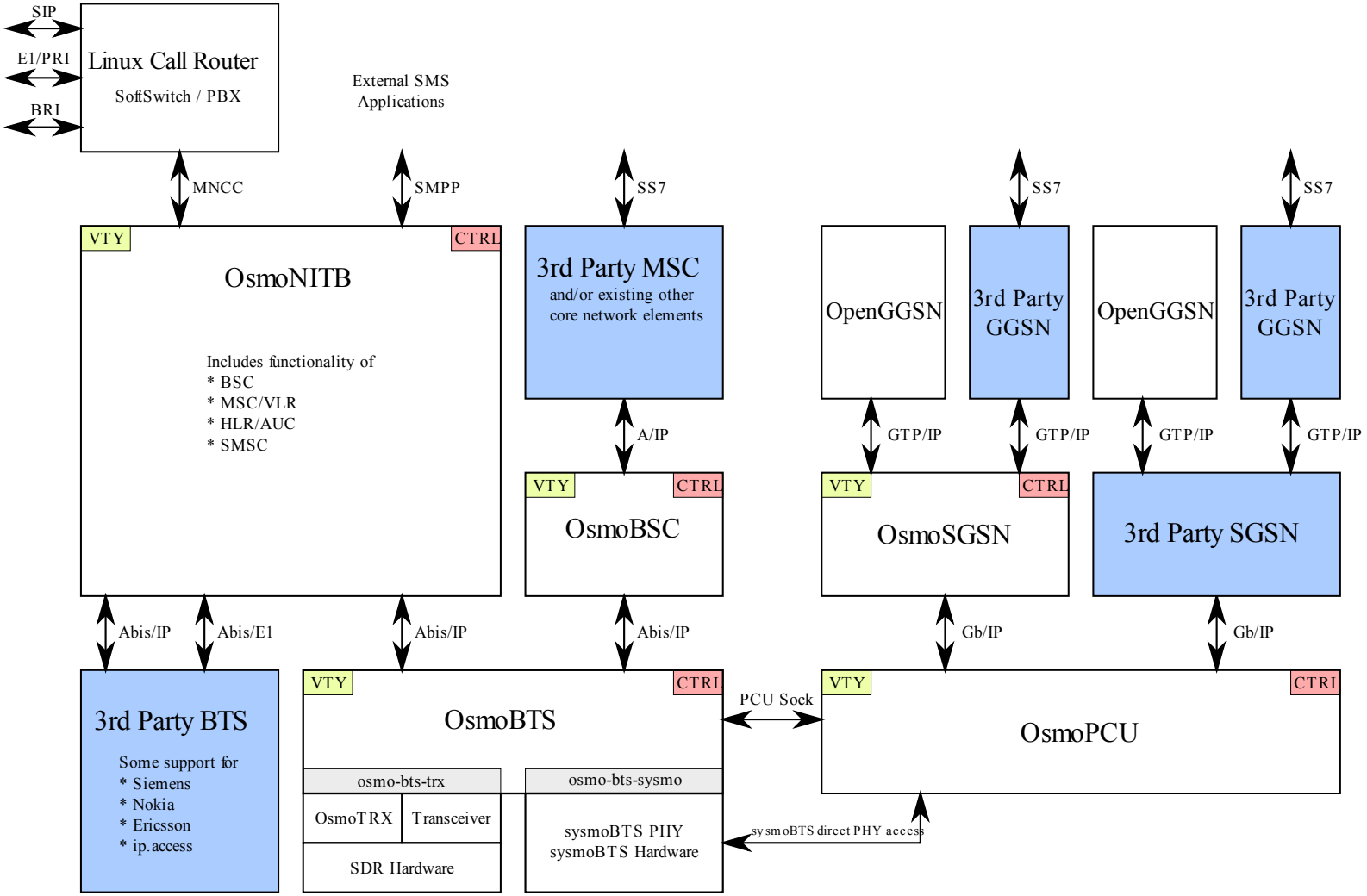
CC

Call Control (voice, circuit switched data, fax)

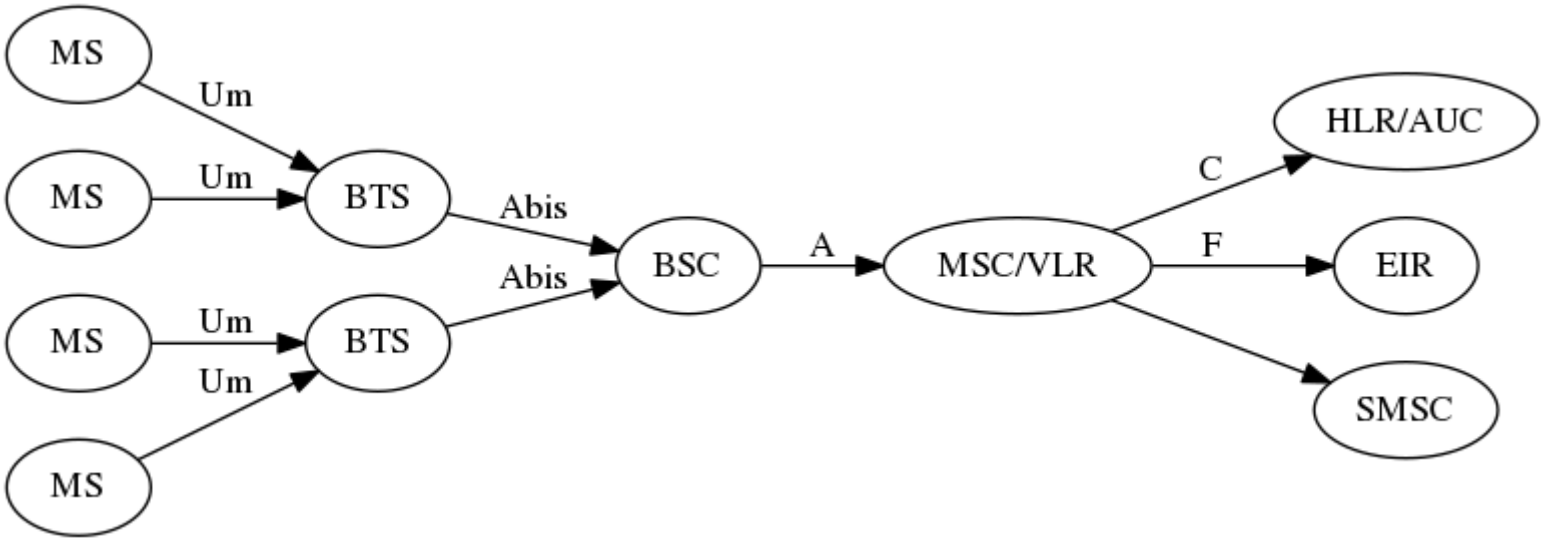
CM

Connection Management

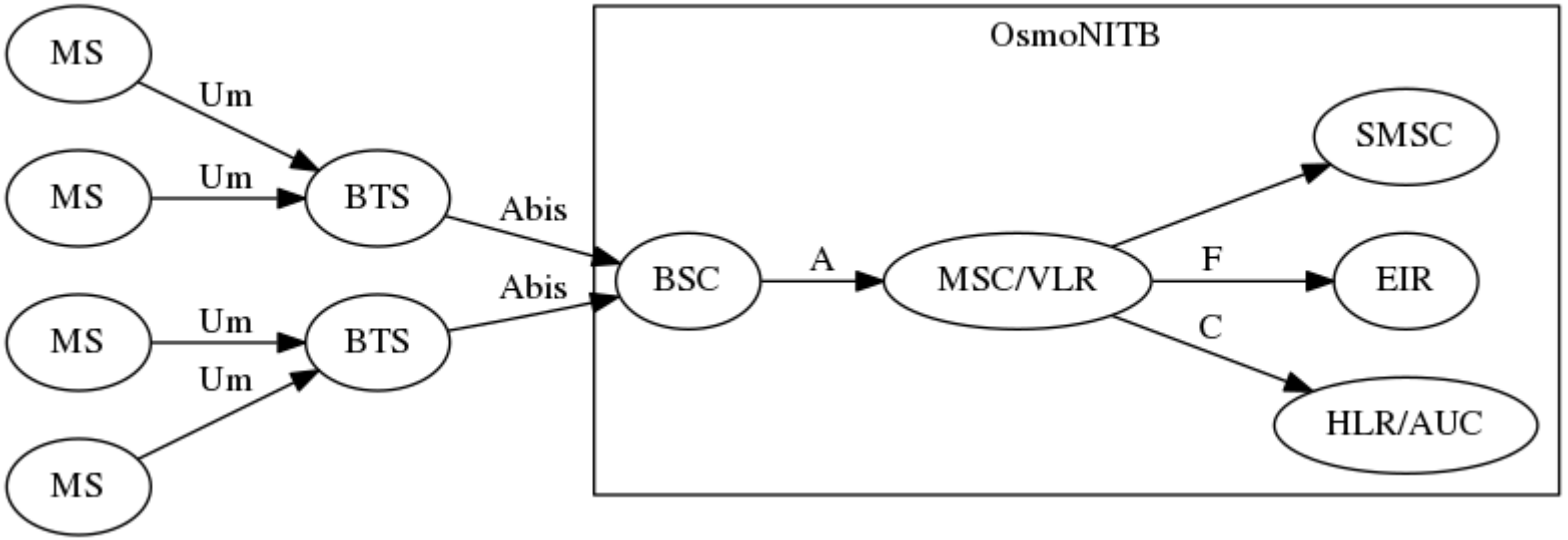
Osmocom GSM components



Classic GSM network as digraph



Simplified OsmoNITB GSM network



which further reduces to the following minimal setup:

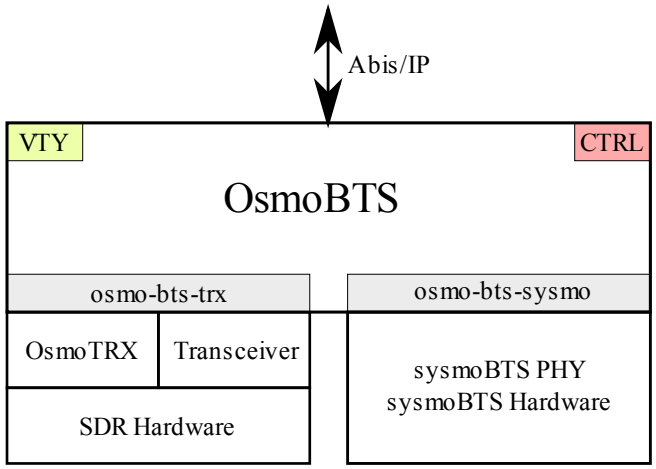
So our minimal setup is a *Phone*, a *BTS* and *OsmoNITB*.

Which BTS to use?

- Proprietary BTS of classic vendor
  - Siemens BS-11 is what we started with
  - Nokia, Ericsson, and others available 2nd hand
- *OsmoBTS* software implementation, running with
  - Proprietary HW + PHY (DSP): *sysmoBTS*, or
  - General purpose SDR (like USRP) + *OsmoTRX*

We assume a sysmoBTS in the following tutorial

OsmoBTS Overview



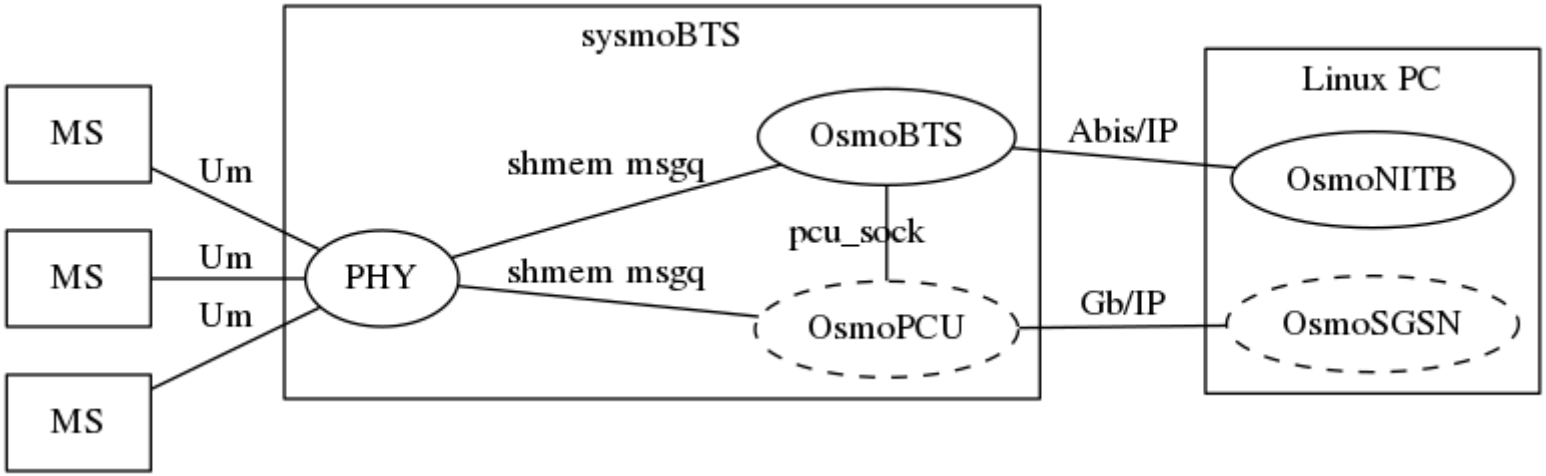
- Implementation of GSM BTS
- supports variety of hardware/PHY options
  - `osmo-bts-sysmo`: BTS family by sysmocom
  - `osmo-bts-trx`: Used with *OsmoTRX* + general-purpose SDR
  - `osmo-bts-octphy`: Octasic OCTBTS hardware / OCTSDR-2G PHY
  - `osmo-bts-litecell15`: Nutaq Litecell 1.5 hardware/PHY

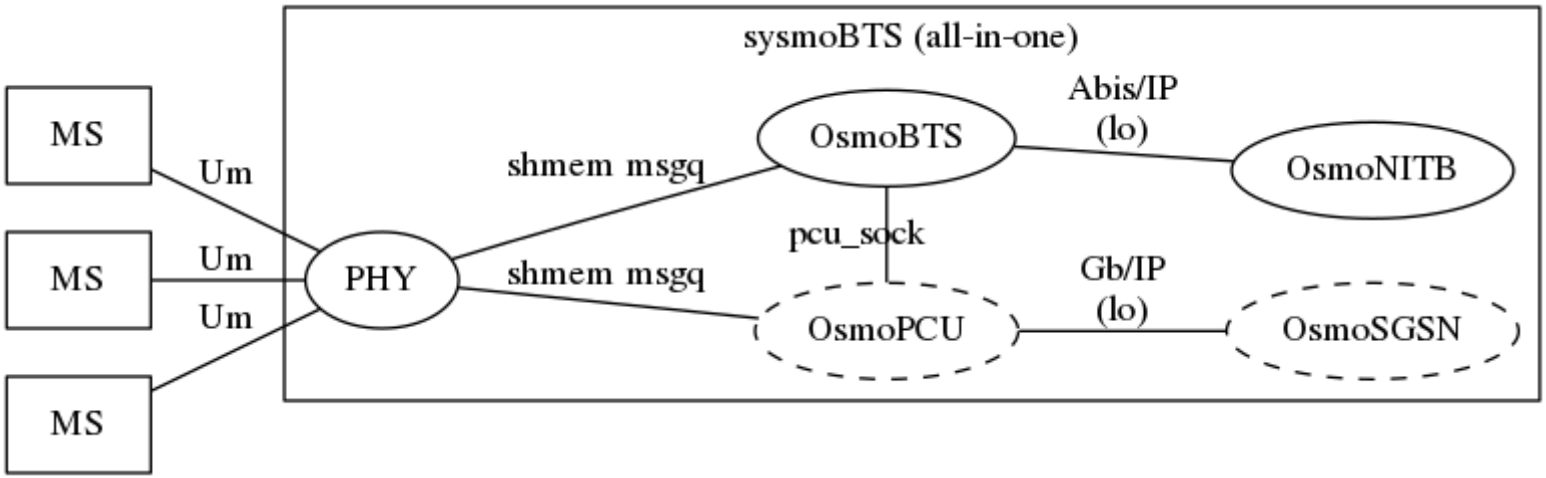
See separate talk about BTS hardware options later today.

BTS Hardware vs. BTS software

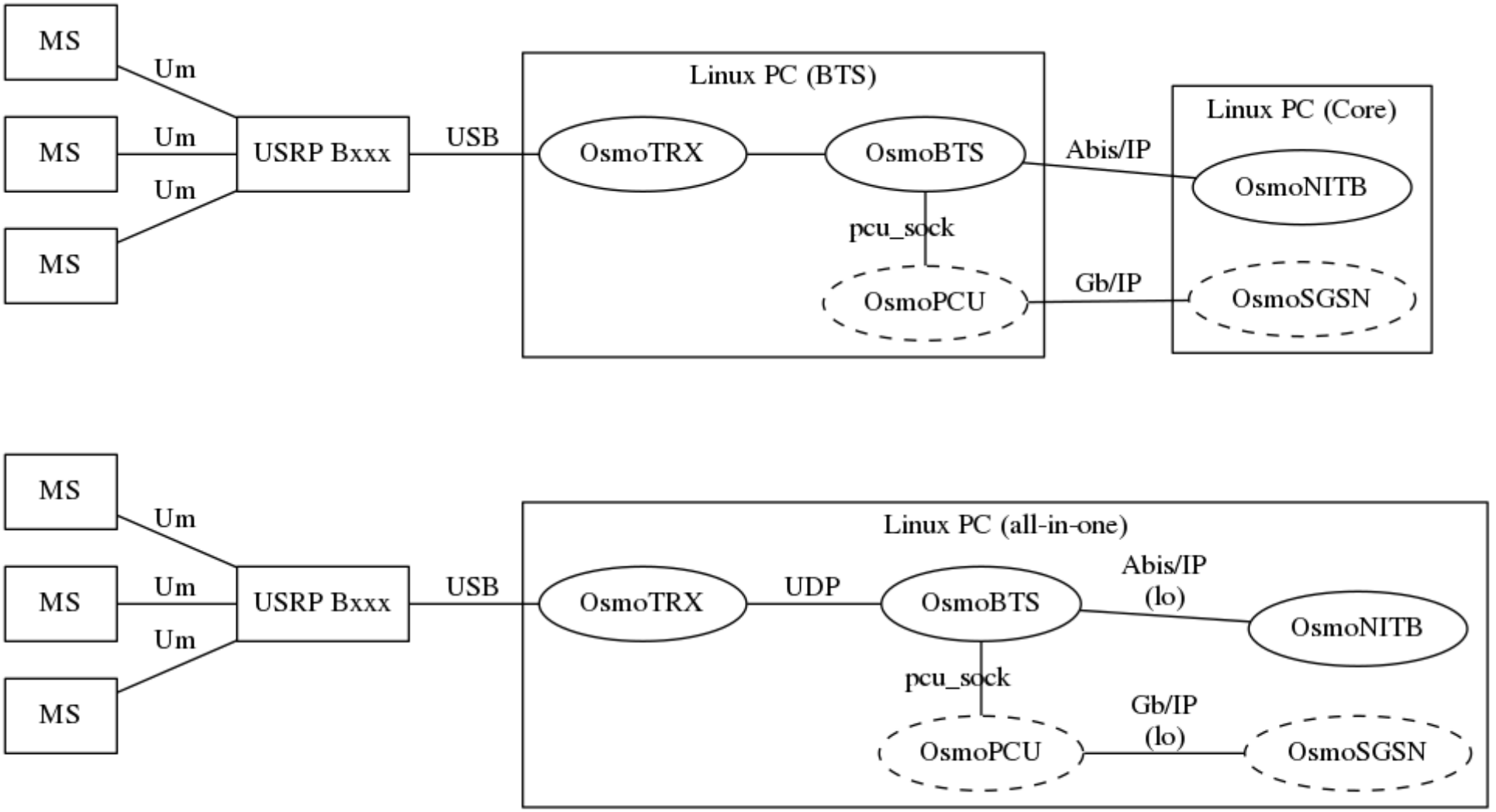
- A classic GSM BTS is hardware + software
- It has two interfaces
  - Um to the radio side, towards phones
  - Abis to the wired back-haul side, towards BSC
- with today's flexible architecture, this is not always true
  - the hardware might just be a network-connected SDR and BTS software runs o a different CPU/computer, *or*
  - the BTS and BSC, or even the NITB may run on the same board

Physical vs. Logical Arch (sysmoBTS)





Physical vs. Logical Arch (SDR e.g. USRP B2xx)



IP layer traffic

- Abis/IP signaling runs inside IPA multiplex inside TCP
  - Port 3002 and 3003 betewen BTS and BSC
  - Connections initiated from BTS to BSC
- Voice data is carried in RTP/UDP on dynamic ports

⇒ Make sure you permit the above communication in your network/firewall config

Configuring Osmocom software

- all *native* Osmo* GSM infrastructure programs share common architecture, as defined by various libraries *libosmo{core,gsm,vty,abis,netif,...}*
- part of this is configuration handling
  - interactive configuration via command line interface (**vty**), similar to Cisco routers
  - based on a fork of the VTY code from Zebra/Quagga, now *libosmovty*
- you can manually edit the config file,
- or use `configure terminal` and interactively change it

Configuring OsmoBTS

- *OsmoBTS* in our example scenario runs on the embedded ARM/Linux system inside the *sysmoBTS*
- we access the *sysmoBTS* via serial console or ssh
- we then edit the configuration file `/etc/osmocom/osmo-bts.cfg` as described in the following slide

Configuring OsmoBTS

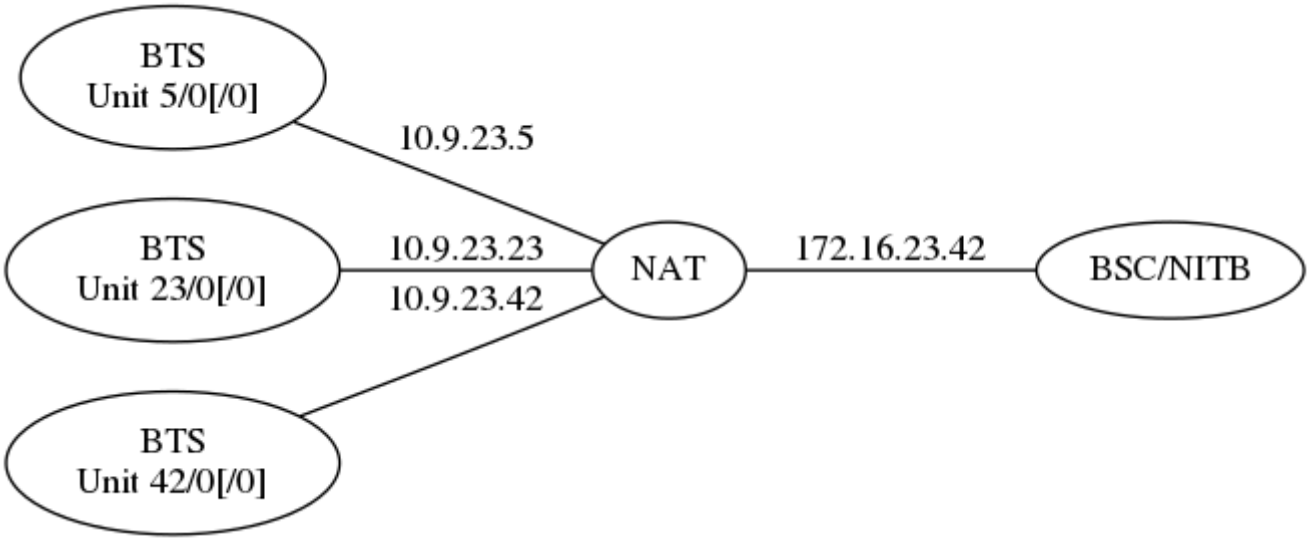
```
bts 0
 band DCS1800 <1>
 ipa unit-id 1801 0 <2>
 oml remote-ip 192.168.100.11 <3>
```

1. the GSM frequency band in which the BTS operates
2. the unit-id by which this BTS identifies itself to the BSC
3. the IP address of the BSC (to establish the OML connection towards it)

Note All other configuration is downloaded by the BSC via OML. So most BTS settings are configured in the BSC/NITB configuration file.

Purpose of Unit ID

- Unit IDs consist of three parts:
  - Site Number, BTS Number, TRX Number



- source IP of all BTSs would be identical

⇒ BSC identifies BTS on Unit ID, not on Source IP!

Configuring OsmoNITB

- *OsmoNITB* is the `osmo-nitb` executable built from the `openbsc` source tree / git repository
  - just your usual `git clone && autoreconf -fi && ./configure && make install`
  - (in reality, the `libosmo*` dependencies are required first...)
  - nightly packages for Debian 8, Ubuntu 16.04 and 16.10 available
- *OsmoNITB* runs on any Linux system, like your speakers' laptop
  - you can actually also run it on the ARM/Linux of the *sysmoBTS* itself, having a literal *Network In The Box* with power as only external dependency

Configuring OsmoNITB

```
network
 network country code 1 <1>
 mobile network code 1 <2>
 short name Osmocom <3>
 long name Osmocom
 auth policy closed <4>
 encryption a5 0 <5>
```

1. MCC (Country Code) e.g. 262 for Germany; 1 == Test
2. MNC (Network Code) e.g. mcc=262, mnc=02 == Vodafone; 1 == Test
3. Operator name to be sent to the phone **after** registration
4. Only accept subscribers (SIM cards) explicitly authorized in HLR
5. Use A5/0 (== no encryption)

Configuring BTS in OsmoNITB (BTS)

```
network
 bts 0
  type sysmobts <1>
  band DCS1800 <2>
  ms max power 33 <3>
  periodic location update 6 <4>
  ip.access unit_id 1801 0 <5>
  codec-support fr hr efr amr <6>
```

1. type of the BTS that we use (must match BTS)
2. frequency band of the BTS (must match BTS)
3. maximum transmit power phones are permitted (33 dBm == 2W)
4. interval at which phones should send periodic location update (6 minutes)
5. Unit ID of the BTS (must match BTS)
6. Voice codecs supported by the BTS

Configuring BTS in OsmoNITB (TRX)

```
network
 bts 0
  trx 0
   arfcn 871 <1>
   max_power_red 0 <2>
   timeslot 0
    phys_chan_config CCCH+SDCCH4 <3>
   timeslot 1
    phys_chan_config TCH/F <4>
    ...
   timeslot 7
    phys_chan_config PDCH <5>
```

1. The RF channel number used by this TRX
2. The maximum power **reduction** in dBm. 0 = no reduction
3. Every BTS needs need one timeslot with a CCCH
4. We configure TS1 to TS6 as TCH/F for voice
5. We configure TS6 as PDCH for GPRS

What a GSM phone does after power-up

- Check SIM card for last cell before switch-off
  - if that cell is found again, use that
  - if not, perform a network scan
    - try to find strong carriers, check if they contain BCCH
    - create a list of available cells + networks
    - if one of the networks MCC+MNC matches first digits of *IMSI*, this is the home network, which has preference over others
- perform *LOCATION UPDATE* (TYPE=IMSI ATTACH) procedure to network
- when network sends *LOCATION UPDATE ACCEPT*, **camp** on that cell

→ let's check if we can perform *LOCATION UPDATE* on our own network

Verifying our network

- look at stderr of *OsmoBTS* and *OsmoNITB*
  - *OsmoBTS* will terminate if Abis cannot be set-up
  - expected to be re-spawned by init / systemd
- use MS to search for networks, try manual registration
- observe registration attempts `logging level mm info`

→ should show *LOCATION UPDATE* request / reject / accept

- use the VTY to explore system state (`show *`)
- use the VTY to change subscriber parameters like extension number

Exploring your GSM networks services

- use `*#100#` from any registered MS to obtain own number
- voice calls from mobile to mobile
- SMS from mobile to mobile
- SMS to/from external applications (via SMPP)
- voice to/from external PBX (via MNCC)
- explore the VTY interfaces of all network elements
  - send SMS from the command line
  - experiment with *silent call* feature
  - experiment with logging levels
- use wireshark to investigate GSM protocols

Using the VTY

- The VTY can be used not only to configure, but also to interactively explore the system status (`show` commands)
- Every Osmo* program has its own telnet port

Program	Telnet Port
OsmoPCU	4240
OsmoBTS	4241
OsmoNITB	4242
OsmoSGSN	4245

- ports are bound to 127.0.0.1 by default
- try tab-completion, `?` and `list` commands

Using the VTY (continued)

- e.g. `show subsciber` to display data about subscriber:

```
OpenBSC> show subscriber imsi 901700000003804
    ID: 12, Authorized: 1
    Extension: 3804
    LAC: 0/0x0
    IMSI: 901700000003804
    TMSI: F2D4FA0A
    Expiration Time: Mon, 07 Dec 2015 09:45:16 +0100
    Paging: not paging Requests: 0
    Use count: 1
```

- try `show bts`, `show trx`, `show lchan`, `show statistics`, ...

Further Reading

User Manuals

See http://ftp.osmocom.org/docs/latest/

Wiki

See http://osmocom.org/projects/openbsc

The End

- so long, and thanks for all the fish
- I hope you have questions!
- have fun exploring mobile technologies using Osmocom
- interested in working with more acronyms? Come join the project!
- Check out https://osmocom.org/ and openbsc@lists.osmocom.org