University of
St Andrews

CS2003: Internet and the Web

Server systems

Server system : purpose

- Overall purpose:
  - **to provide access to resources, and provide application services.**
- Many different types of "service":
  - **Direct interaction with client system.**
  - Support services (control plane, e.g. DNS).
  - Management of resources (management plane, e.g. data centre).
  - Servers can also provide services to other servers.
  - Many technologies, e.g. virtualisation, containers, cloud, …
- **Function and form of service depends on type of application and nature of interaction with client (user).**

Server functions

- Access to the application functions and resources:
  - Well-defined interface via protocol operations.
  - Many control and security features, e.g. authentication access control, user accounts and user-specific configuration, etc.
- **Check and implement operations requested by client:**
  - includes access control and security features.
- Check and interpret **network communication**.
- Report system and network **events** and **errors**:
  - To the remote user via client (using agreed protocol).
  - To the system administrators and management applications, e.g. local logging (event logs, security logs, error logs etc).

Scalability and Performance

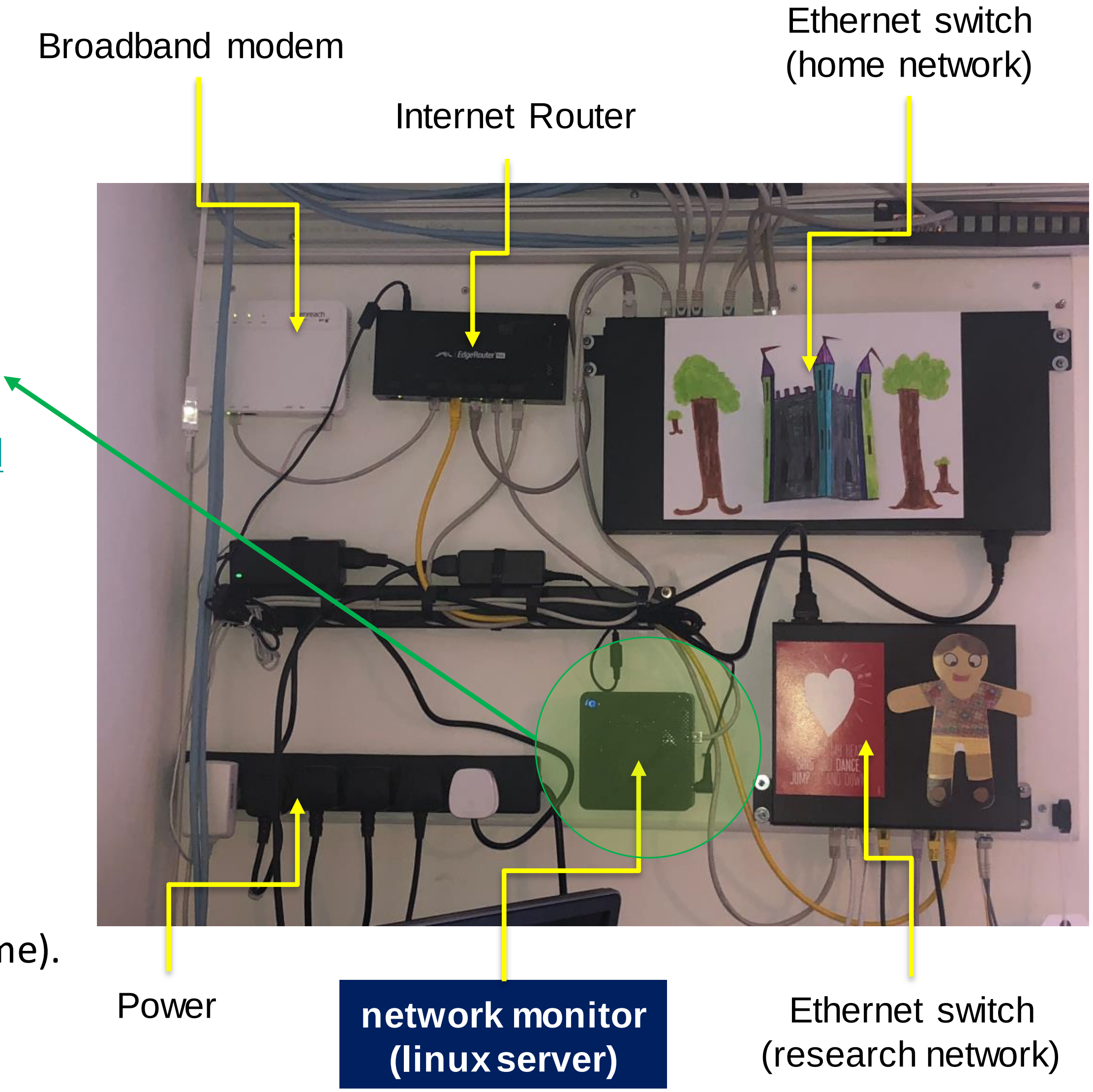
- Major distinctions between client system and server: **scalability** and **performance**.
- **Scalability**:
  - Service and resources for many users, not just one.
  - Simultaneous access for many users.
- **Performance**:
  - Provide a satisfactory service for each user.
  - Performance impact for individual users should be minimal as service is scaled.

Servers : scaling (1)

https://www.msi.com/Desktop/Cubi-N.html

Intel Celeron CPU N3060, 1.60GHz, 2 cores
4GB RAM
128 GB HD
~ £130
~12cm × ~11cm × ~4cm, ~500g.
40W (0.04 KW) maximum.
Provides network monitoring services for a household of users. 1 administrator (part time).

Typical load: 20% - 30%



Broadband modem
Internet Router
Ethernet switch (home network)
Power
network monitor (linux server)
Ethernet switch (research network)

Servers : scaling (2)

https://www.google.com/about/datacenters/locations/st-ghislain/

(As of 04 Oct 2019 – much is estimated.)

Many server grade CPUs, 8+ cores, 3GHz+ per server (10s of thousands of cores).
Multi-TB RAM overall.
Multi-TB storage overall.

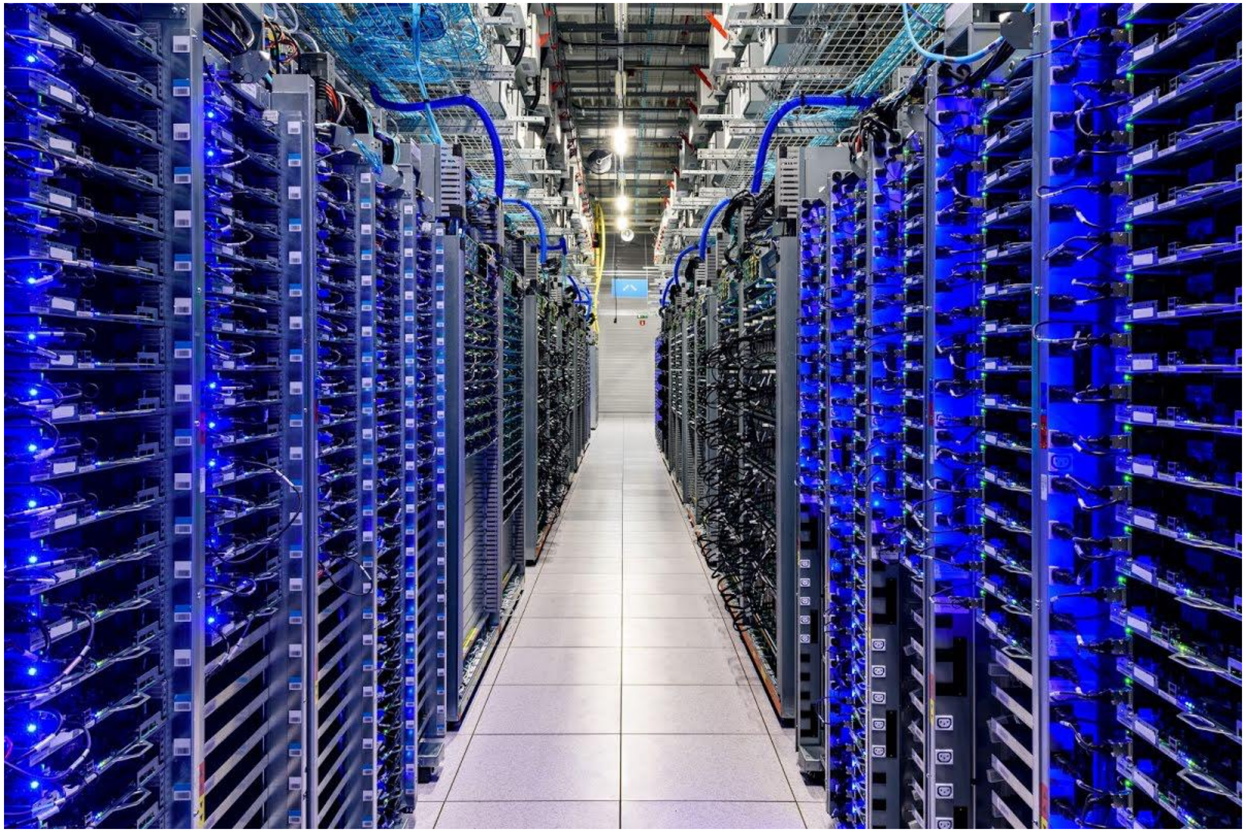
€1.6 billion, invested so far.

Size of village / small town (estimated).

~1GW+ (estimated).

Provides application services for millions of users. ~350 full-time staff.

Typical load: ~70%+ (estimated)



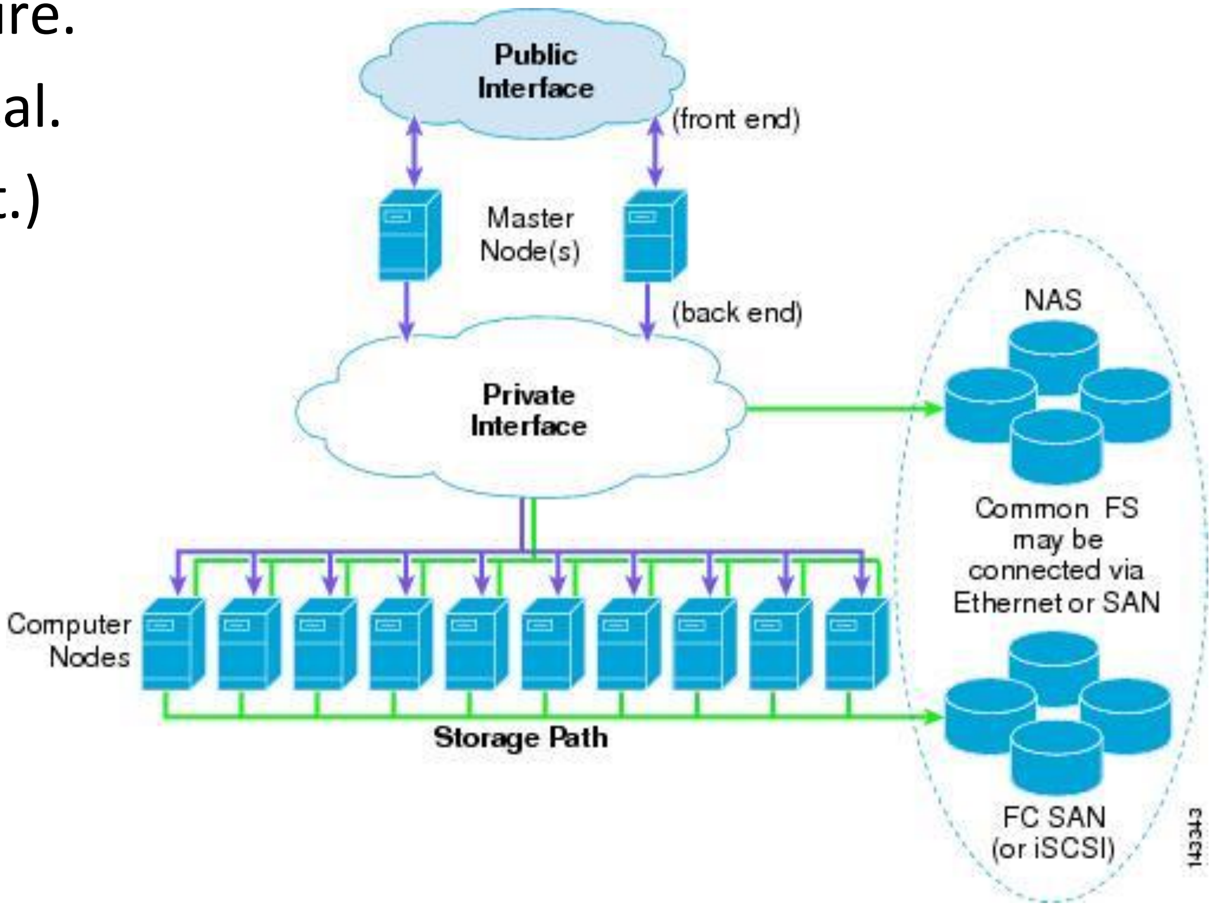
https://www.google.com/about/datacenters/gallery/
Google's datacentre in St. Ghislain, Belgium.

Services and server software design (1)

- Design of server software:
  - Scalability and Performance (and other non-functional requirements).
- The software has to be allow many simultaneous users, and many simultaneous operations / tasks.
- All servers in a datacentre might not be identical:
  - Specific tasks to provide overall service.
  - Specific hardware and software for specific tasks.

Services and server software design (2)

- Example from Cisco:
  - No “standard” architecture.
  - Distributed system, typical.
  - (Exact detail not relevant.)
- Some servers provide **front-end** services:
  - Customer facing.
  - **Queue** of requests.
- Many servers provide **back-end** services.
  - Resources not directly accessible to customers.



https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCInfra_1.html

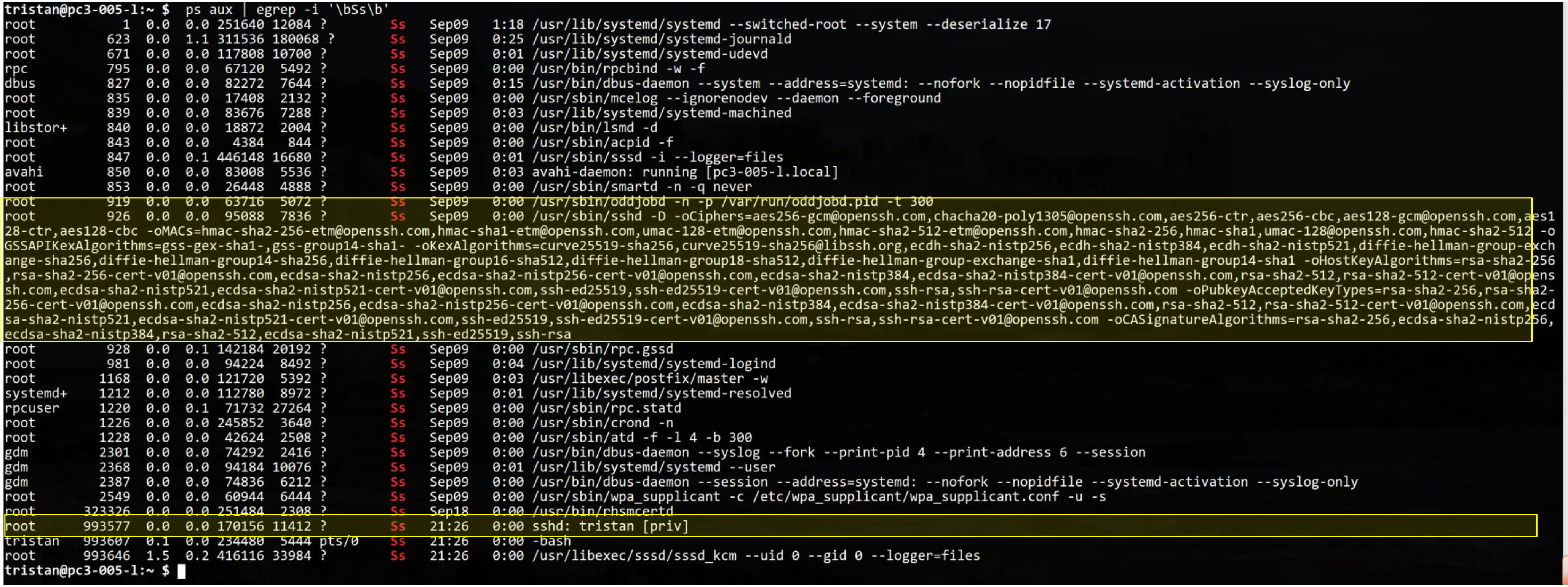
Examples on Linux (1)

- Servers run as “background” processes, often as the root user, with privileged access to resources.
- On Linux, executing:

  ```
  ps auxw | egrep -i '\bSs\b'
  ```

  will give a list of (many) running server processes.
- CS lab machines are configured as client systems:
  - relatively few different servers.
  - many **local service** applications running as root.

Examples on Linux (2)



```
tristan@pc3-005-l:~ $  ps aux | egrep -i '\bSs\b'
root           1  0.0  0.0 251640 12084 ?        Ss   Sep09   1:18 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
root         623  0.0  1.1 311536 180068 ?       Ss   Sep09   0:25 /usr/lib/systemd/systemd-journald
root         671  0.0  0.0 117808 10700 ?        Ss   Sep09   0:01 /usr/lib/systemd/systemd-udevd
rpc          795  0.0  0.0  67120  5492 ?        Ss   Sep09   0:00 /usr/bin/rpcbind -w -f
dbus         827  0.0  0.0  82272  7644 ?        Ss   Sep09   0:15 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
root         835  0.0  0.0  17408  2132 ?        Ss   Sep09   0:00 /usr/sbin/mcelog --ignorenodev --daemon --foreground
root         839  0.0  0.0  83676  7288 ?        Ss   Sep09   0:03 /usr/lib/systemd/systemd-machined
libstor+     840  0.0  0.0  18872  2004 ?        Ss   Sep09   0:00 /usr/bin/lsmd -d
root         843  0.0  0.0   4384   844 ?        Ss   Sep09   0:00 /usr/sbin/acpid -f
root         847  0.0  0.1 446148 16680 ?        Ss   Sep09   0:01 /usr/sbin/sssd -i --logger=files
avahi        850  0.0  0.0  83008  5536 ?        Ss   Sep09   0:03 avahi-daemon: running [pc3-005-l.local]
root         853  0.0  0.0  26448  4888 ?        Ss   Sep09   0:00 /usr/sbin/smartd -n -q never
root         919  0.0  0.0  63716  5072 ?        Ss   Sep09   0:00 /usr/sbin/oddjobd -n -p /var/run/oddjobd.pid -t 300
root         926  0.0  0.0  95088  7836 ?        Ss   Sep09   0:00 /usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com,chacha20-poly1305@openssh.com,aes256-ctr,aes256-cbc,aes128-gcm@openssh.com,aes1
28-ctr,aes128-cbc -oMACs=hmac-sha2-256-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha2-256,hmac-sha1,umac-128@openssh.com,hmac-sha2-512 -o
GSSAPIKexAlgorithms=gss-gex-sha1-,gss-group14-sha1- -oKexAlgorithms=curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exch
ange-sha256,diffie-hellman-group14-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha1 -oHostKeyAlgorithms=rsa-sha2-256
,rsa-sha2-256-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384,ecdsa-sha2-nistp384-cert-v01@openssh.com,rsa-sha2-512,rsa-sha2-512-cert-v01@opens
sh.com,ecdsa-sha2-nistp521,ecdsa-sha2-nistp521-cert-v01@openssh.com,ssh-ed25519,ssh-ed25519-cert-v01@openssh.com,ssh-rsa,ssh-rsa-cert-v01@openssh.com -oPubkeyAcceptedKeyTypes=rsa-sha2-256,rsa-sha2-
256-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384,ecdsa-sha2-nistp384-cert-v01@openssh.com,rsa-sha2-512,rsa-sha2-512-cert-v01@openssh.com,ecd
sa-sha2-nistp521,ecdsa-sha2-nistp521-cert-v01@openssh.com,ssh-ed25519,ssh-ed25519-cert-v01@openssh.com,ssh-rsa,ssh-rsa-cert-v01@openssh.com -oCASignatureAlgorithms=rsa-sha2-256,ecdsa-sha2-nistp256,
ecdsa-sha2-nistp384,rsa-sha2-512,ecdsa-sha2-nistp521,ssh-ed25519,ssh-rsa
root         928  0.0  0.1 142184 20192 ?        Ss   Sep09   0:00 /usr/sbin/rpc.gssd
root         981  0.0  0.0  94224  8492 ?        Ss   Sep09   0:04 /usr/lib/systemd/systemd-logind
root        1168  0.0  0.0 121720  5392 ?        Ss   Sep09   0:03 /usr/libexec/postfix/master -w
systemd+    1212  0.0  0.0 112780  8972 ?        Ss   Sep09   0:01 /usr/lib/systemd/systemd-resolved
rpcuser     1220  0.0  0.1  71732 27264 ?        Ss   Sep09   0:00 /usr/sbin/rpc.statd
root        1226  0.0  0.0 245852  3640 ?        Ss   Sep09   0:00 /usr/sbin/crond -n
root        1228  0.0  0.0  42624  2508 ?        Ss   Sep09   0:00 /usr/sbin/atd -f -l 4 -b 300
gdm         2301  0.0  0.0  74292  2416 ?        Ss   Sep09   0:00 /usr/bin/dbus-daemon --syslog --fork --print-pid 4 --print-address 6 --session
gdm         2368  0.0  0.0  94184 10076 ?        Ss   Sep09   0:01 /usr/lib/systemd/systemd --user
gdm         2387  0.0  0.0  74836  6212 ?        Ss   Sep09   0:00 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
root        2549  0.0  0.0  60944  6444 ?        Ss   Sep09   0:00 /usr/sbin/wpa_supplicant -c /etc/wpa_supplicant/wpa_supplicant.conf -u -s
root      323326  0.0  0.0 251484  2308 ?        Ss   Sep18   0:00 /usr/bin/rhsmcertd
root      993577  0.0  0.0 170156 11412 ?        Ss   21:26   0:00 sshd: tristan [priv]
tristan   993607  0.1  0.0 234480  5444 pts/0    Ss   21:26   0:00 -bash
root      993646  1.5  0.2 416116 33984 ?        Ss   21:26   0:00 /usr/libexec/sssd/sssd_kcm --uid 0 --gid 0 --logger=files
tristan@pc3-005-l:~ $ █
```

All servers use only local filestore and local resources.

Some servers (e.g. sshd) will spawn additional processes.

Our scope for CS2003

- Single server, multiple clients (~30 users max).
- Server can deal with multiple users / requests:
  - Can use **queues** for requests (client and server).
  - Can use **threads** for tasks (client and server).
  - No use of background / daemon processes.
- Some flavour of considerations in large-scale systems engineering for datacentres:
  - Large-scale services/servers beyond scope of CS2003 😐

FIFO queue (1)

- First In First Out (FIFO):
  - aka First Come First Served (FCFS)
- Requests can be queued at a server:
  - A *task* takes incoming requests and queues them.
  - One or more other *tasks* process the request.
- Lots of queuing / scheduling possibilities for real servers systems, e.g. in datacentres:
  - Tends to be commercially sensitive information.
  - FIFO is a simple example only.

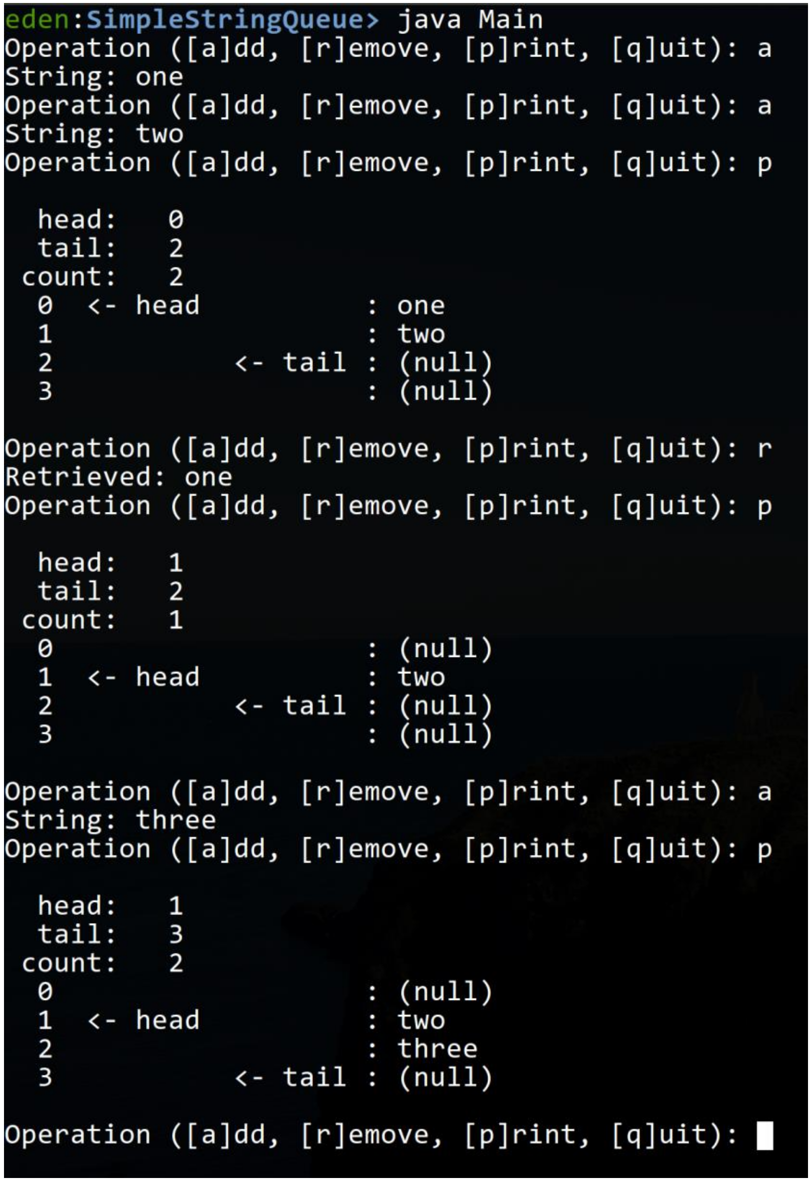
FIFO queue (2)

- Simple FIFO queue:
  - Use an array for holding requests / data (strings).
  - **tail**: where the next request gets queued.
  - **head**: the first request / data to be processed.
- Maximum size of queue:
  - Queue can become full: requests can not be added.
- **Circular FIFO**:
  - Management of head / tail allows a “logical circuit”.

FIFO queue (3)

- CS2003/Examples/CS2003-Examples-wk05/SimpleStringQueue/
  - SimpleStringQueue, QueueEmptyException, QueueFullException
- Example of FIFO circular queue - Main.java:
  - simple program to use the queue.
  - [a]dd strings to the queue.
  - [r]emove strings.
  - [p]rint, shows head, tail and contents of queue.

FIFO queue (4)



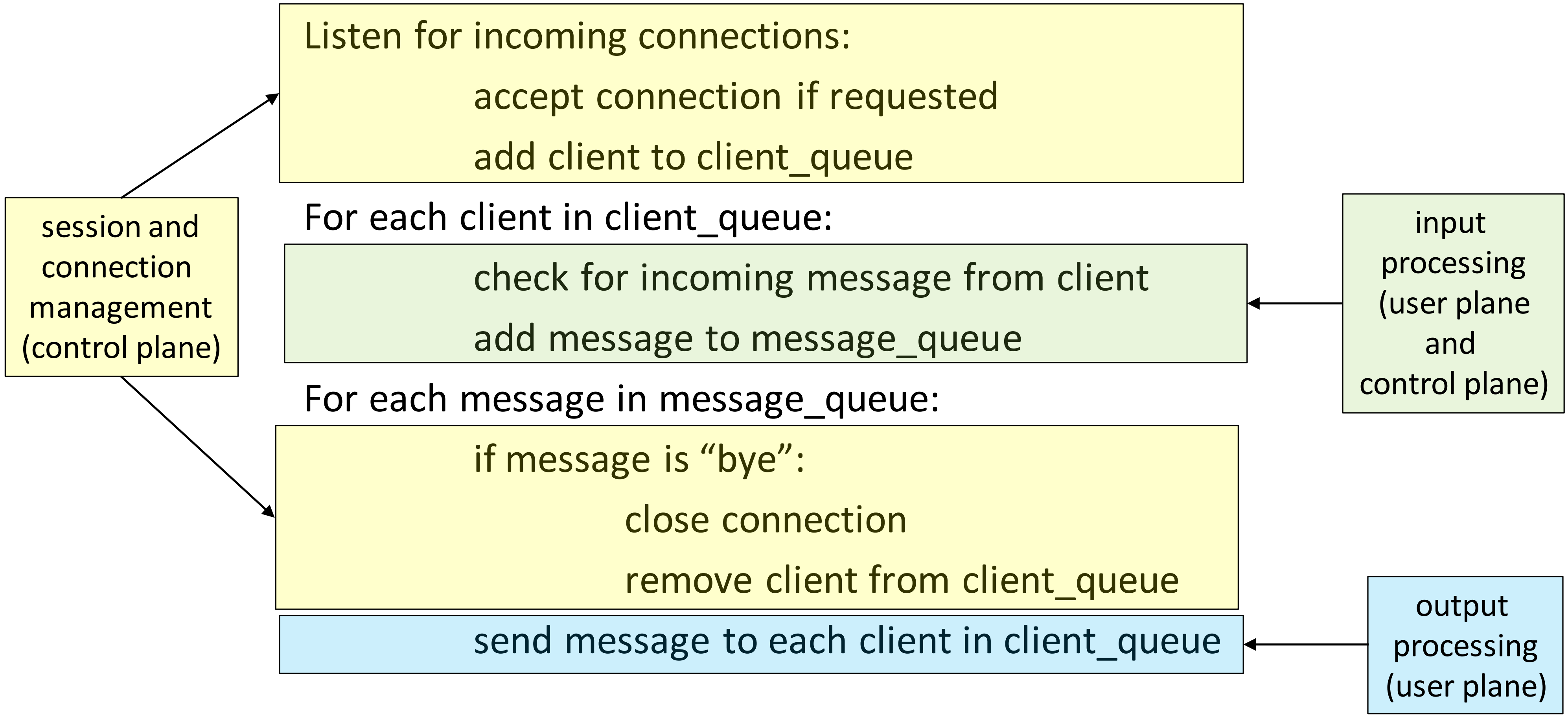
```
eden:SimpleStringQueue> java Main
Operation ([a]dd, [r]emove, [p]rint, [q]uit): a
String: one
Operation ([a]dd, [r]emove, [p]rint, [q]uit): a
String: two
Operation ([a]dd, [r]emove, [p]rint, [q]uit): p

  head:   0
  tail:   2
 count:   2
  0  <- head          : one
  1                   : two
  2          <- tail : (null)
  3                   : (null)

Operation ([a]dd, [r]emove, [p]rint, [q]uit): r
Retrieved: one
Operation ([a]dd, [r]emove, [p]rint, [q]uit): p

  head:   1
  tail:   2
 count:   1
  0                   : (null)
  1  <- head          : two
  2          <- tail : (null)
  3                   : (null)

Operation ([a]dd, [r]emove, [p]rint, [q]uit): a
String: three
Operation ([a]dd, [r]emove, [p]rint, [q]uit): p

  head:   1
  tail:   3
 count:   2
  0                   : (null)
  1  <- head          : two
  2                   : three
  3          <- tail : (null)

Operation ([a]dd, [r]emove, [p]rint, [q]uit): █
```

Example multi-user server: MultiChat

- TCP-based server:
  - Allows multiple clients to connect.
  - Simple, text-based application protocol.
  - Simple, session control (“bye” message).
  - Incoming messages relayed to all all connected clients.
- Queues for clients and messages.
- Logging with LogFileWriter (from wk03).
- Static configuration in files for simplicity:
  - Could use Configuration (from wk03).

Multi-user server: MultiChatServer (1)

MultiChatServer server loop:



```
Listen for incoming connections:
        accept connection if requested
        add client to client_queue
For each client in client_queue:
        check for incoming message from client
        add message to message_queue
For each message in message_queue:
        if message is “bye”:
                close connection
                remove client from client_queue
        send message to each client in client_queue
```

session and connection management (control plane)

input processing (user plane and control plane)

output processing (user plane)

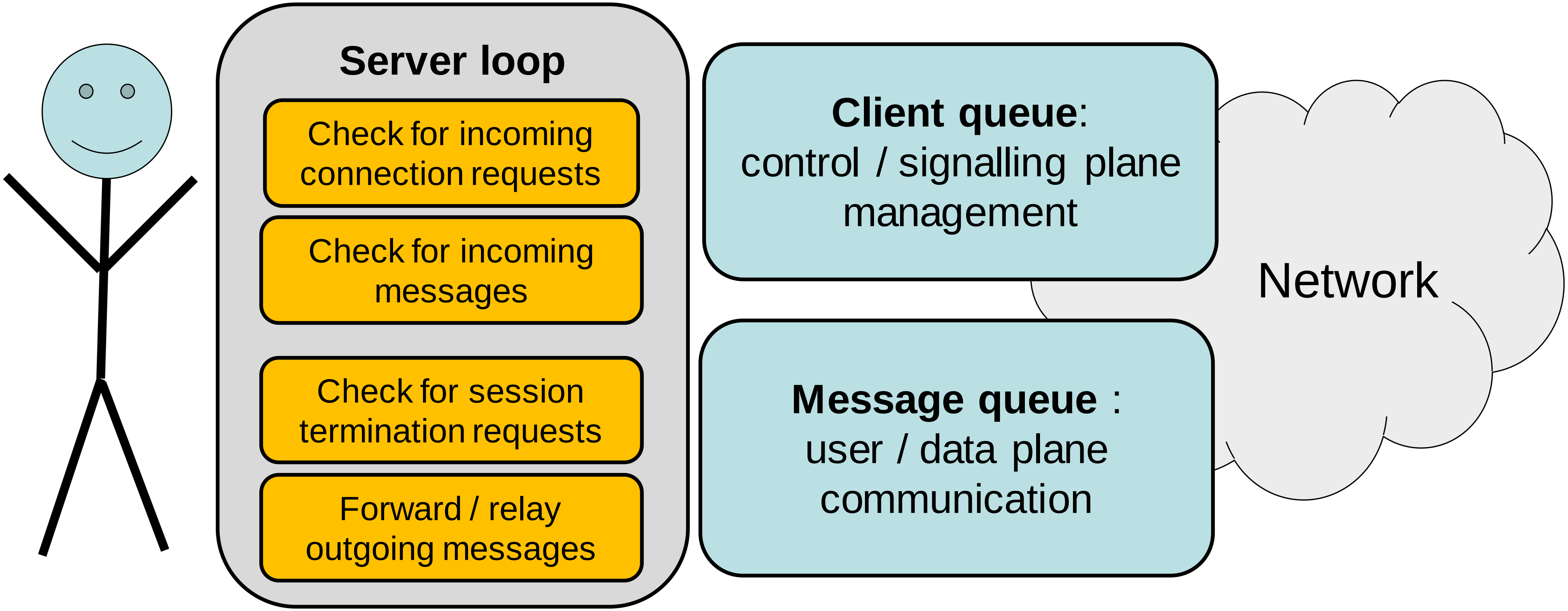
Multi-user server: MultiChatServer (2)

- CS2003/Examples/CS2003-Examples-wk05/MultiChat/
- Also general version of FIFO queue:
  - SimpleObjectQueue
- ChatMessage:
  - Simple application protocol, printable strings, use of regular expressions for decoding / parsing messages.
  - ChatMessageEncodeException, ChatMessageDecodeException
  - ChatMessage_Test, testing for encode/decode.

MultiChatClient

- Text-based client:
  - Servername and portnumber as arguments, so you can connect to each others servers.
  - Use of queues for demonstration purposes.
- Client loop:
  - Check keyboard, add string to transmission queue:
    - » If “bye”, mark as “finished” for client.
  - Check network, put messages in receive queue.
  - Transmit anything in transmission queue.
  - Print to screen anything in receive queue.

Summary


Server loop
Check for incoming connection requests
Check for incoming messages
Check for session termination requests
Forward / relay outgoing messages
Client queue: control / signalling plane management
Message queue : user / data plane communication
Network