Mapping a service-oriented architecture

Mapping a service-oriented architecture

Kappa architecture

machine learning

big-data

Mapping a microservices architecture with Docker and SDNs

NoSQL

timeseries

Metallica

Mapping your infrastructure

Peter Bourgon

☞ Harmen Bus

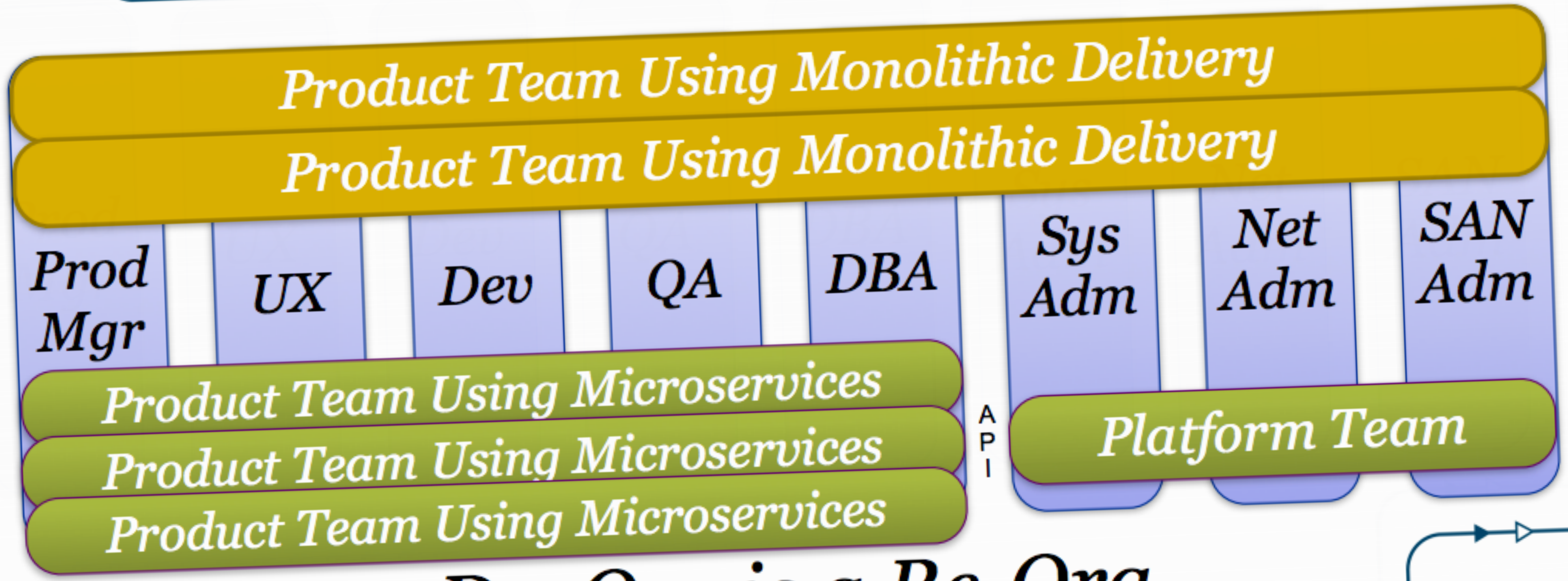
☞ David Kaltschmidt

1. **Motivation**
2. **What we want**
3. **How to build it**

Motivation

A dev/ops world

Breaking Down the SILOs


Product Team Using Monolithic Delivery
Product Team Using Monolithic Delivery
Prod Mgr
UX
Dev
QA
DBA
Sys Adm
Net Adm
SAN Adm
Product Team Using Microservices
Product Team Using Microservices
Product Team Using Microservices
API
Platform Team

DevOps is a Re-Org

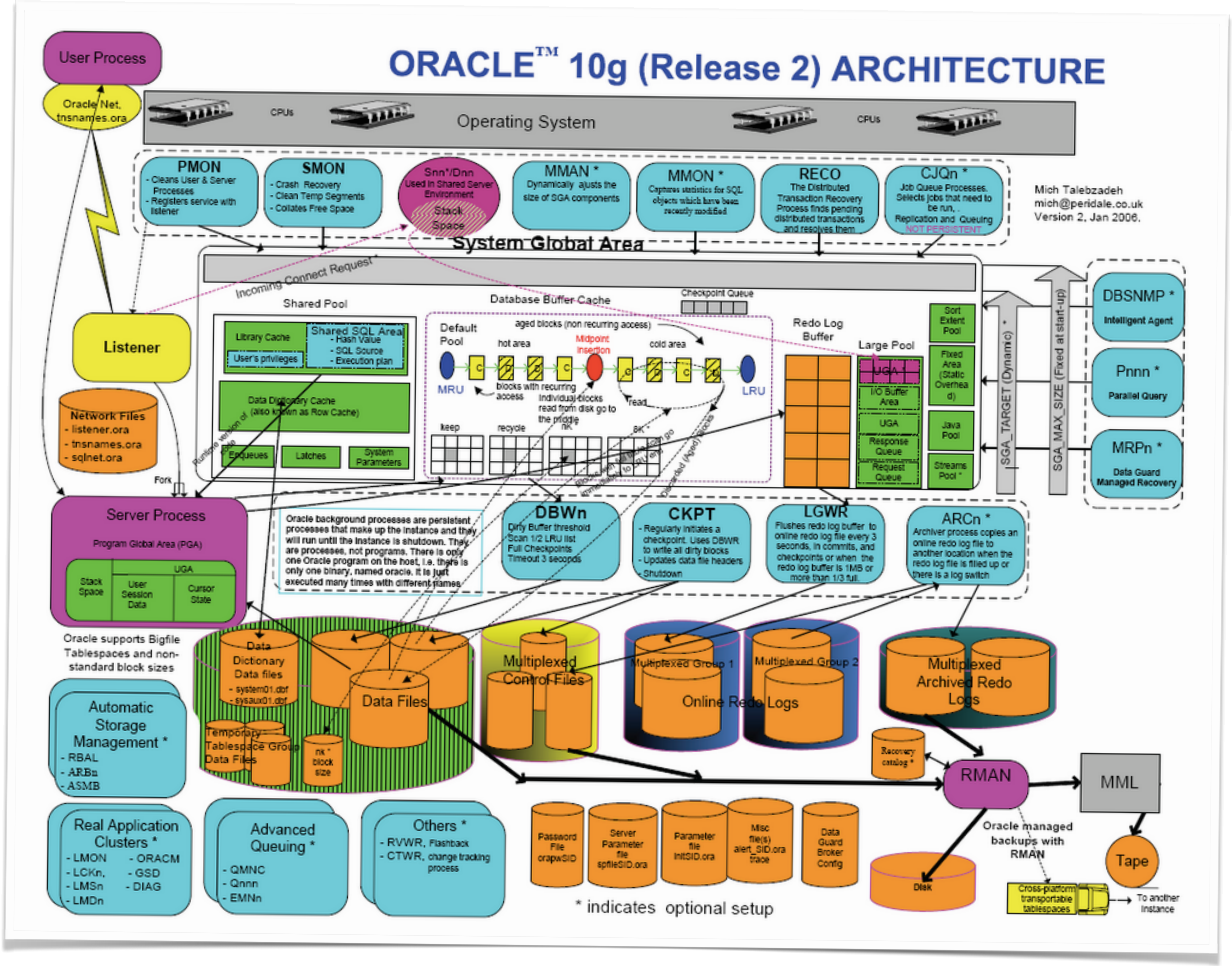
Speed = **good**

What I learned from my time at Netflix

- *Speed wins in the marketplace*

Speed = **dangerous**

ORACLE™ 10g (Release 2) ARCHITECTURE

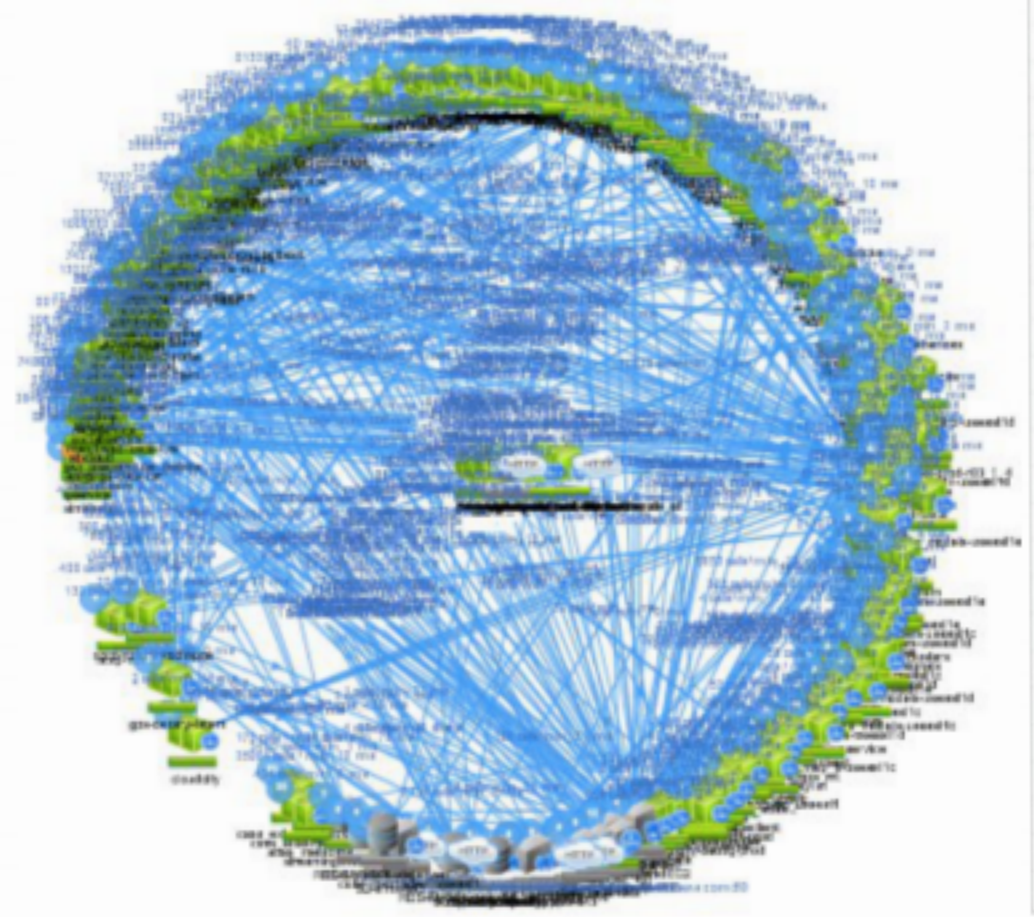


User Process
Oracle Net, tnsnames.ora
CPUs
Operating System
CPUs
PMON
- Cleans User & Server Processes
- Registers service with listener
SMON
- Crash Recovery
- Clean Temp Segments
- Collates Free Space
Snn*/Dnn
Used in Shared Server Environment
Stack Space
MMAN *
Dynamically ajusts the size of SGA components
MMON *
Captures statistics for SQL objects which have been recently modified
RECO
The Distributed Transaction Recovery Process finds pending distributed transactions and resolves them
CJQn *
Job Queue Processes. Selects jobs that need to be run, . Replication and Queuing
NOT PERSISTENT
Mich Talebzadeh
mich@peridale.co.uk
Version 2, Jan 2006.
System Global Area
Incoming Connect Request*
Listener
Network Files
- listener.ora
- tnsnames.ora
- sqlnet.ora
Fork
Shared Pool
Library Cache
User's privileges
Shared SQL Area
- Hash Value
- SQL Source
- Execution plan
Data Dictionary Cache (also known as Row Cache)
Runtime version of code
Enqueues
Latches
System Parameters
Database Buffer Cache
Checkpoint Queue
Default Pool
aged blocks (non recurring access)
hot area
Midpoint Insertion
cold area
MRU
LRU
blocks with recurring access
individual blocks read from disk go to the middle
read
keep
recycle
nK *
8K *
Blocks with full table scan go immediately to LRU end
Discarded (Aged) blocks
Redo Log Buffer
Large Pool
UGA *
I/O Buffer Area
UGA
Response Queue
Request Queue
Sort Extent Pool
Fixed Area (Static Overhead)
Java Pool
Streams Pool *
SGA_TARGET (Dynamic) *
SGA_MAX_SIZE (Fixed at start-up)
DBSNMP *
Intelligent Agent
Pnnn *
Parallel Query
MRPn *
Data Guard Managed Recovery
Server Process
Program Global Area (PGA)
Stack Space
UGA
User Session Data
Cursor State
DBWn
Dirty Buffer threshold
Scan 1/2 LRU list
Full Checkpoints
Timeout 3 seconds
CKPT
- Regularly initiates a checkpoint. Uses DBWR to write all dirty blocks
- Updates data file headers
- Shutdown
LGWR
Flushes redo log buffer to online redo log file every 3 seconds, in commits, and checkpoints or when the redo log buffer is 1MB or more than 1/3 full.
ARCn *
Archiver process copies an online redo log file to another location when the redo log file is filled up or there is a log switch
Oracle supports Bigfile Tablespaces and non-standard block sizes
Data Dictionary Data files
- system01.dbf
- sysaux01.dbf
Data Files
Temporary Tablespace Group Data Files
nk * block size
Multiplexed Control Files
Multiplexed Group 1
Multiplexed Group 2
Online Redo Logs
Multiplexed Archived Redo Logs
Recovery catalog *
RMAN
MML
Tape
Oracle managed backups with RMAN
Disk
Cross-platform transportable tablespaces
To another instance
Automatic Storage Management *
- RBAL
- ARBn
- ASMB
Real Application Clusters *
- LMON
- LCKn,
- LMSn
- LMDn
- ORACM
- GSD
- DIAG
Advanced Queuing *
- QMNC
- Qnnn
- EMNn
Others *
- RVWR, Flashback
- CTWR, change tracking process
Password File orapwSID
Server Parameter file spfileSID.ora
Parameter file initSID.ora
Misc file(s) alert_SID.ora trace
Data Guard Broker Config

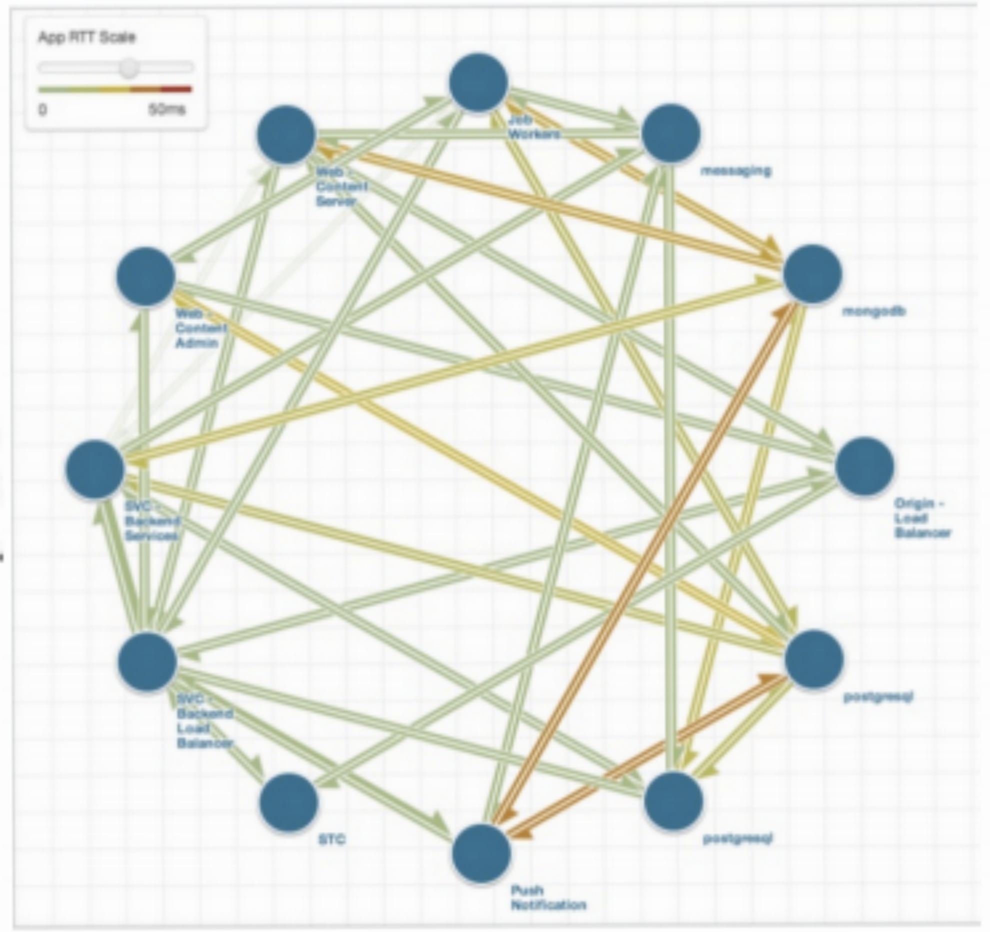
Oracle background processes are persistent processes that make up the instance and they will run until the instance is shutdown. They are processes, not programs. There is only one Oracle program on the host, i.e. there is only one binary, named oracle. It is just executed many times with different names

* indicates optional setup

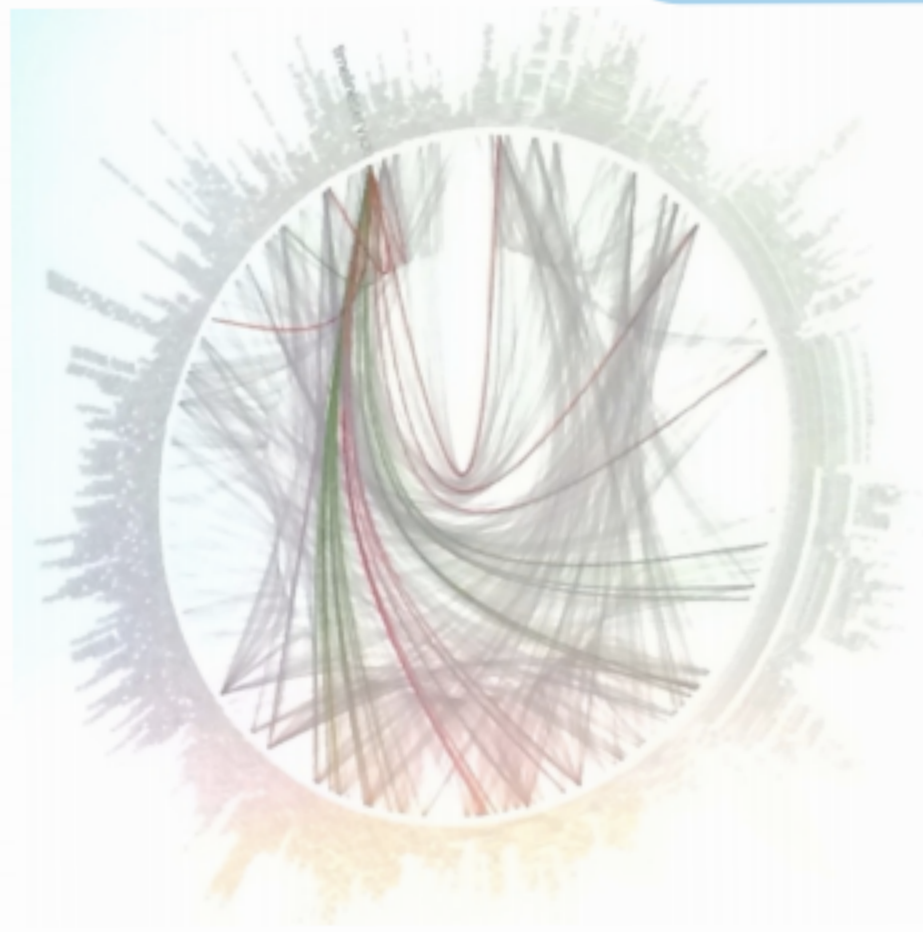
“Death Star” Architecture Diagrams



Netflix



Gilt Groupe (12 of 450)



Twitter

Invariant:
Complexity is unavoidable

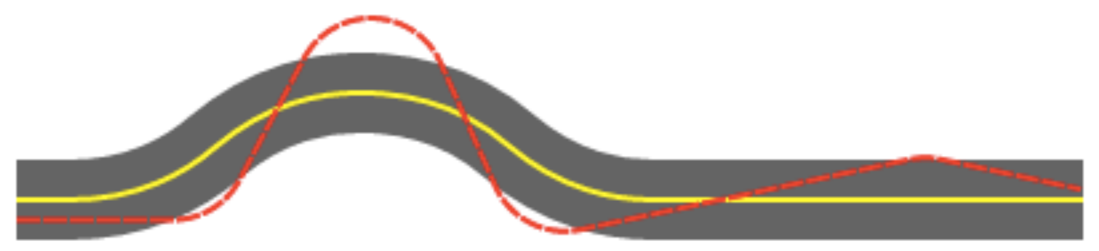
What we want

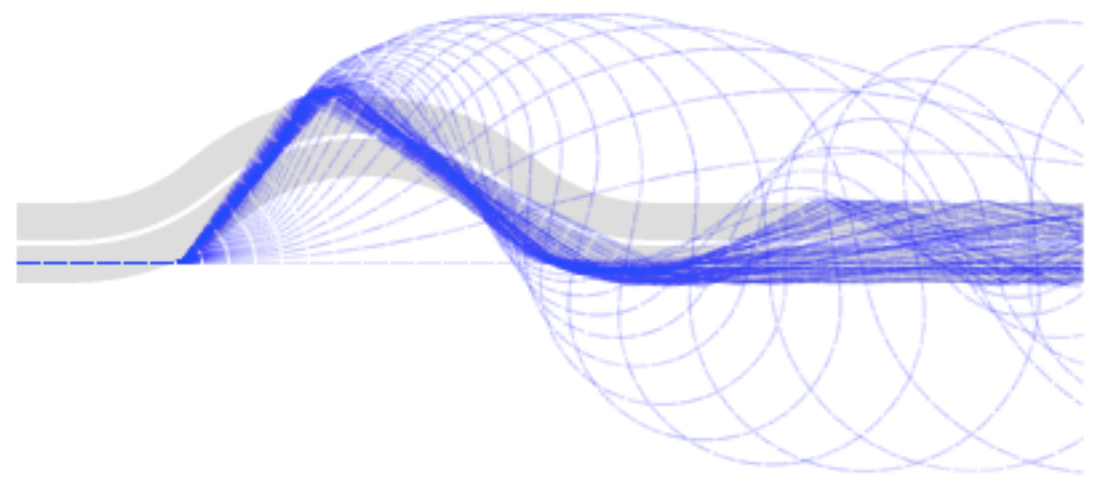
Make complexity
understandable

Visual, dynamic, humane

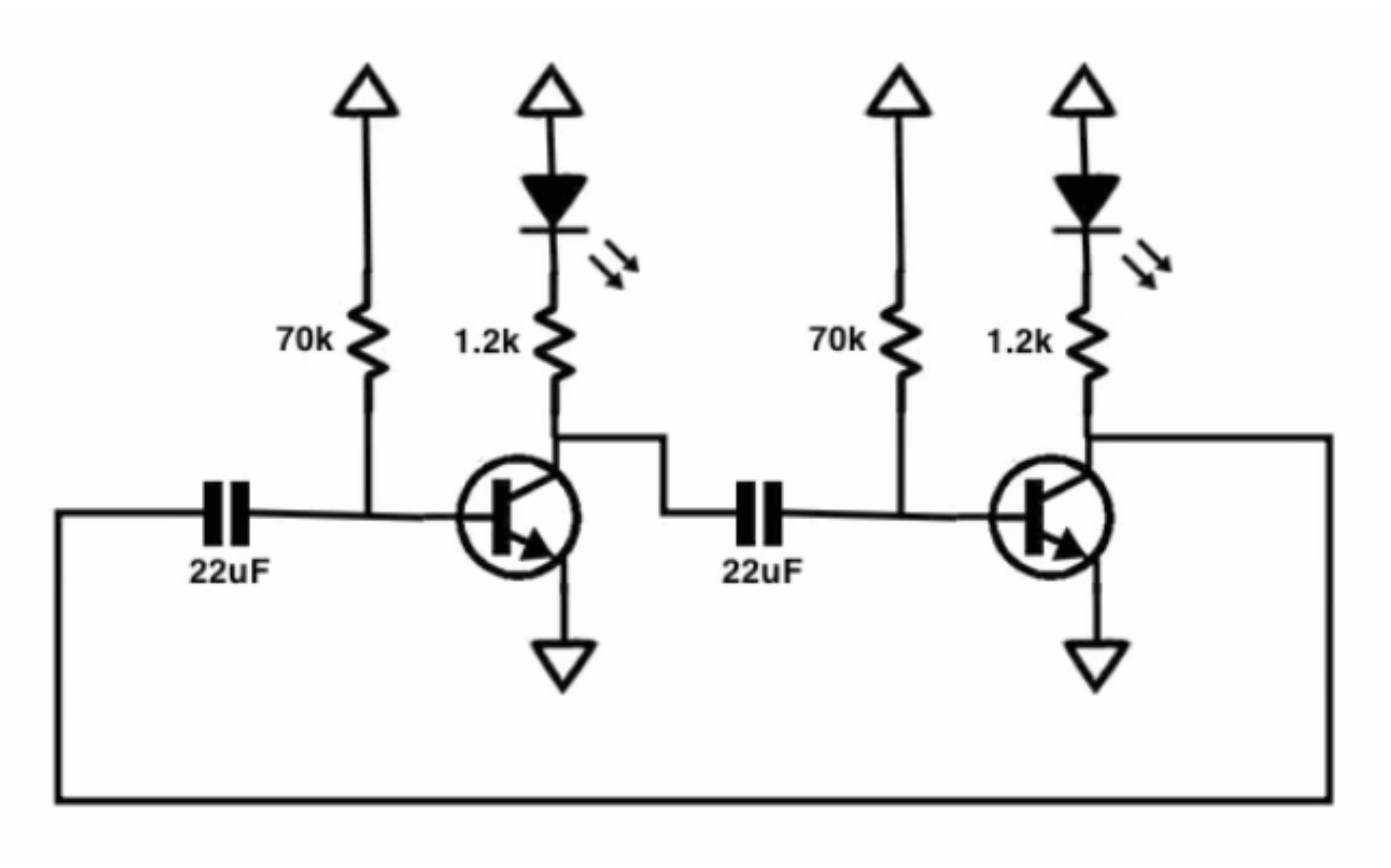
Visual · http://worrydream.com/LadderOfAbstraction/
Dynamic · https://vimeo.com/66085662
Humane · http://worrydream.com/TheHumaneRepresentationOfThoughtTalk

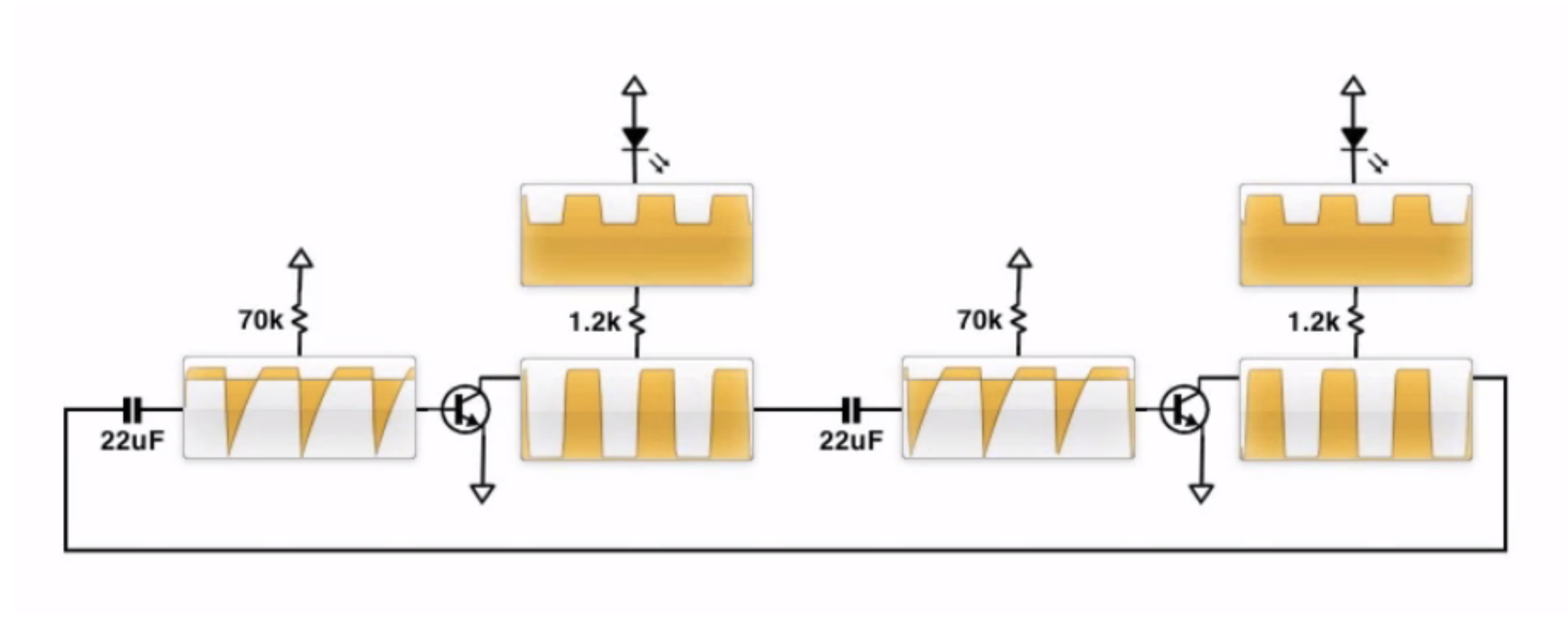
Visual



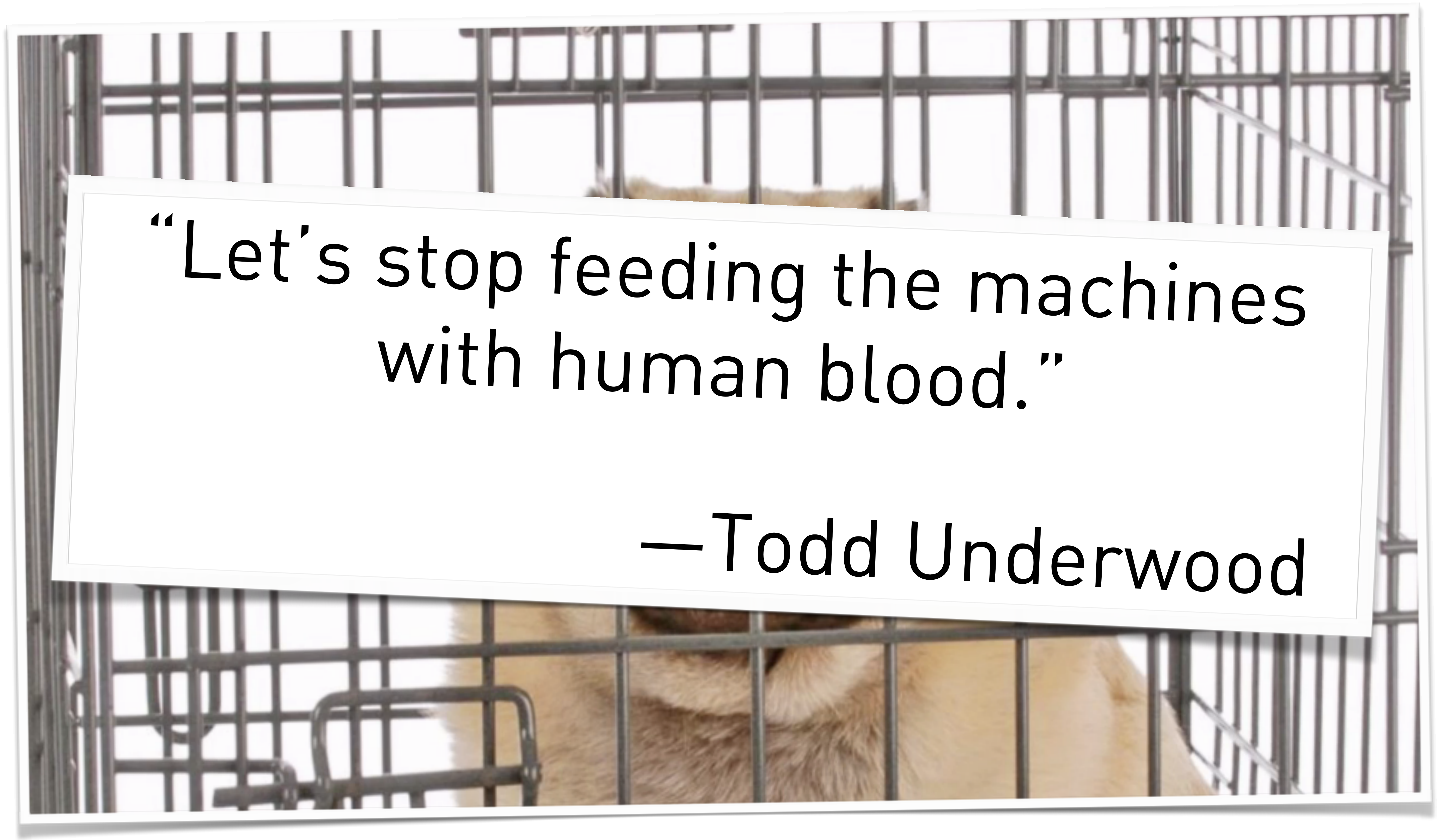


Dynamic





Humane



“Let’s stop feeding the machines with human blood.”

—Todd Underwood

Visual = **graphical**
Dynamic = **responsive**
Humane = **no config**

Invariant:

Model as a directed graph

(visual)

Invariant:

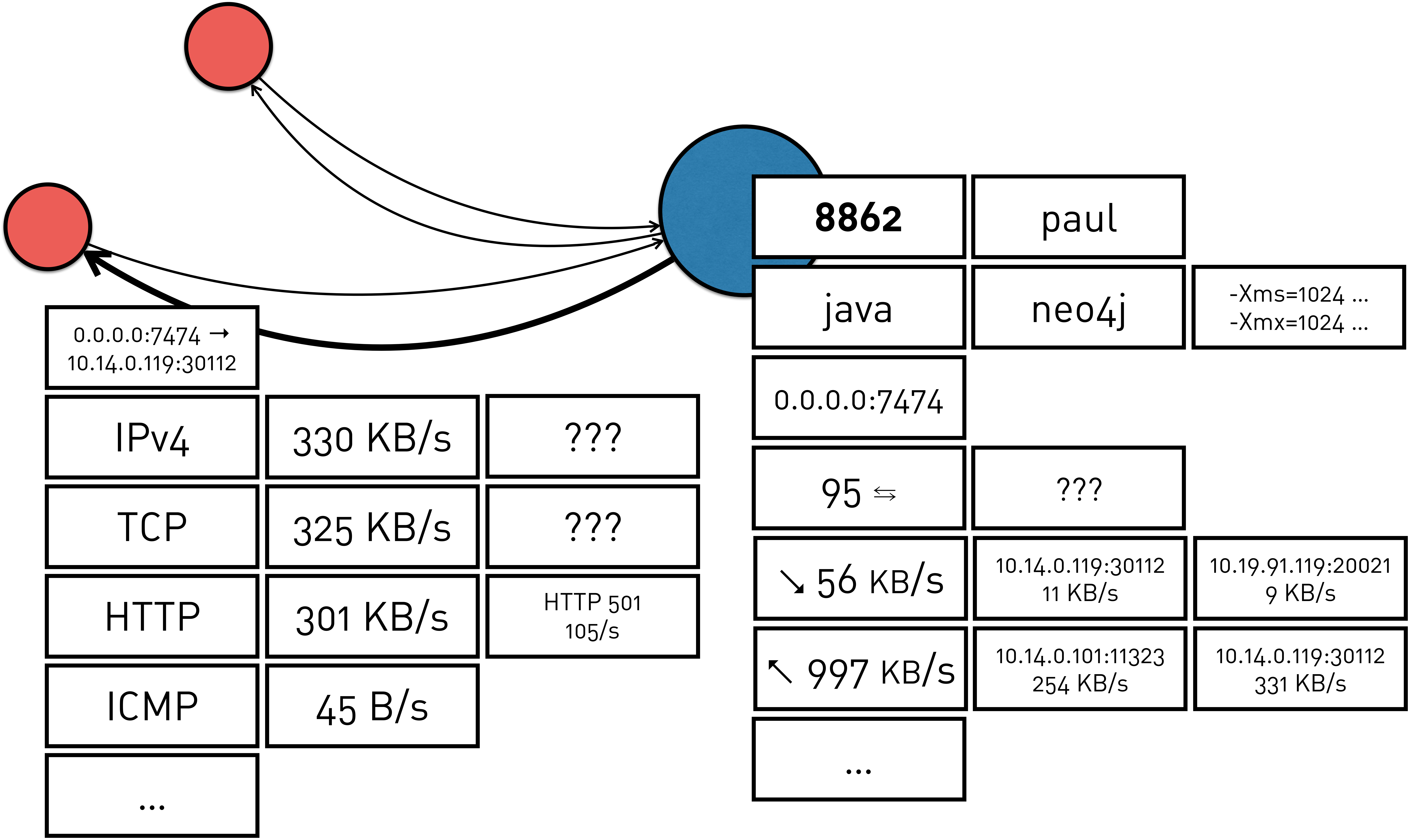
An instantaneous, updating view

(dynamic)

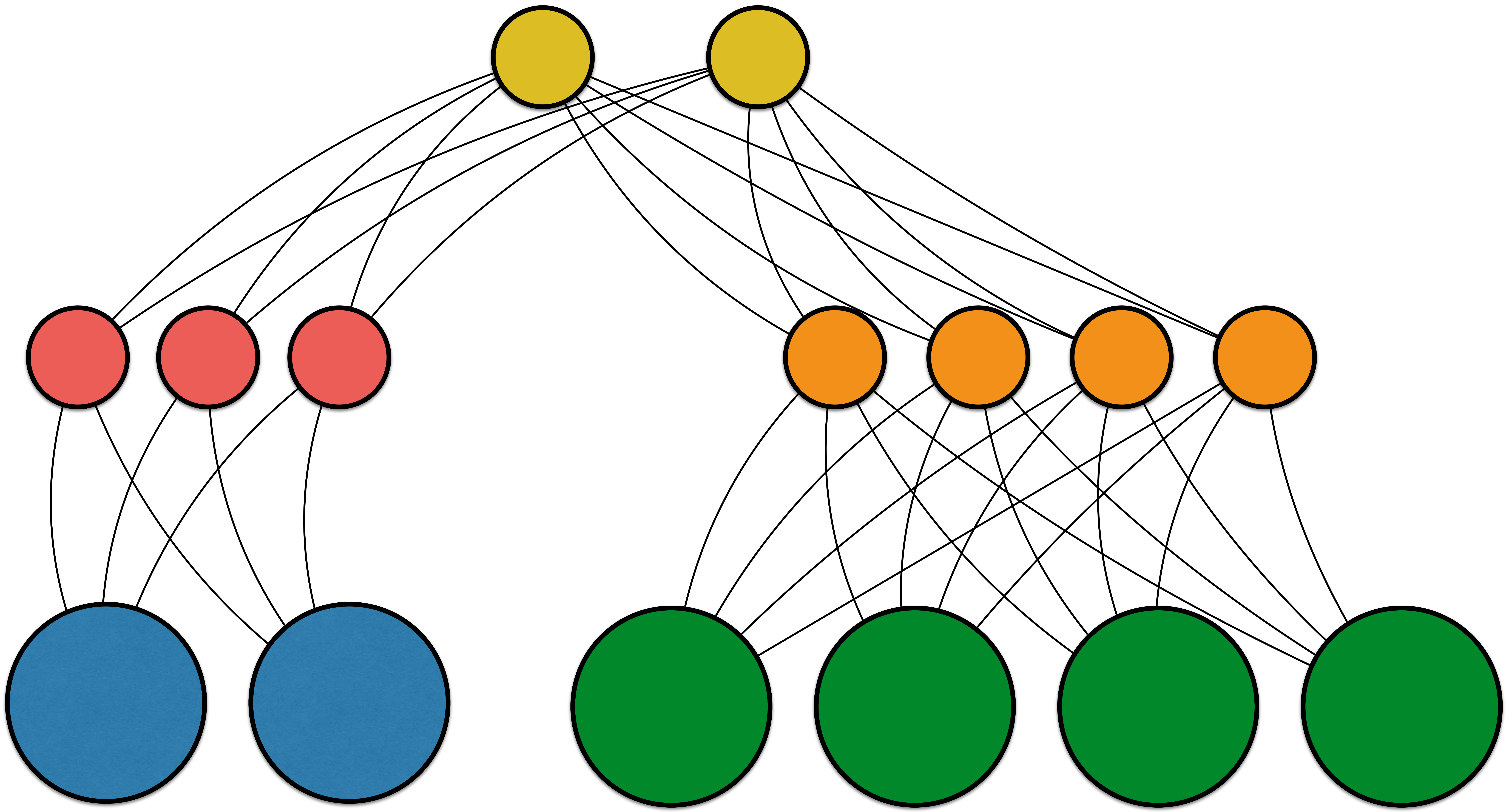
Invariant:

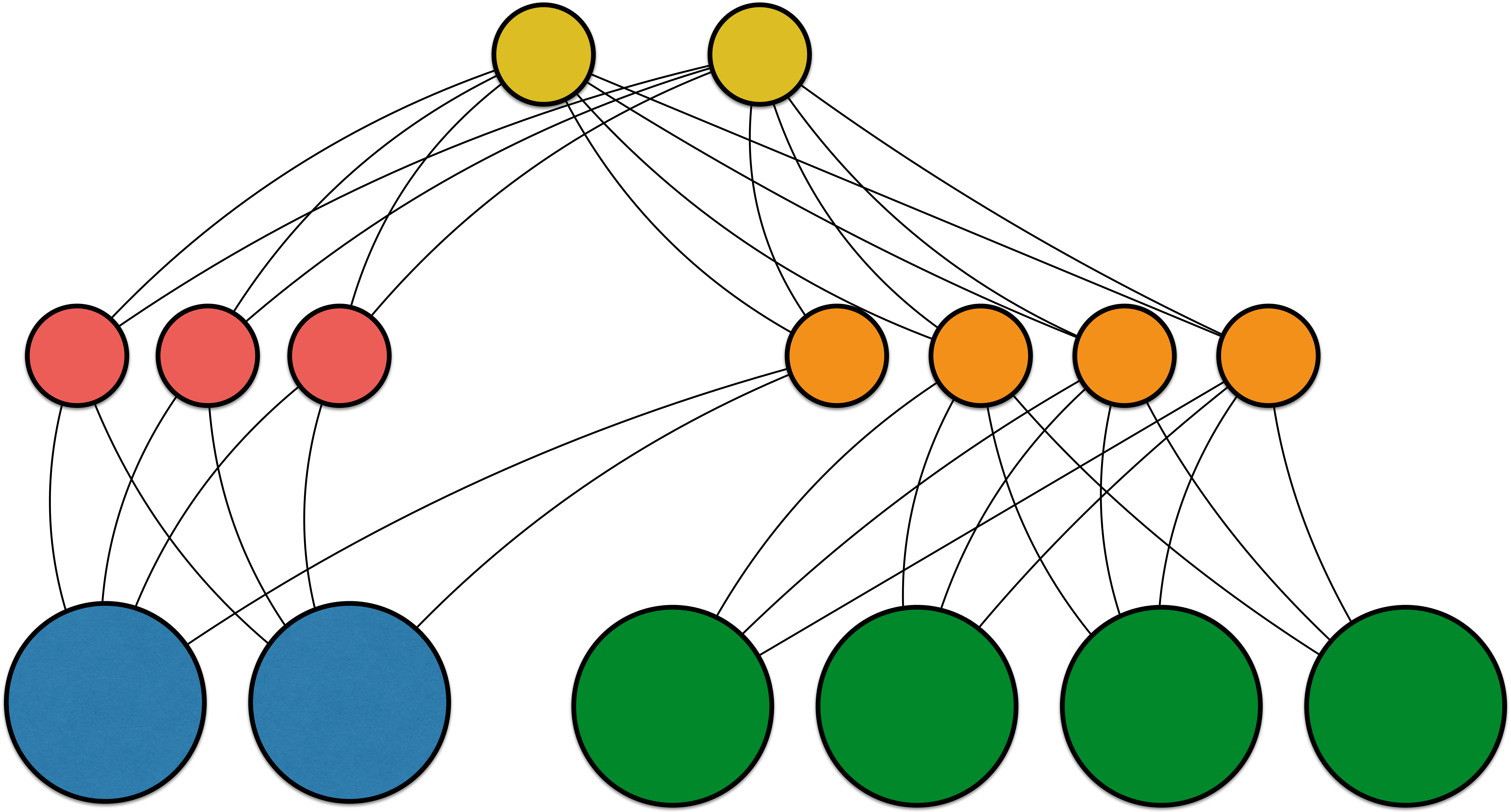
No configuration or declaration

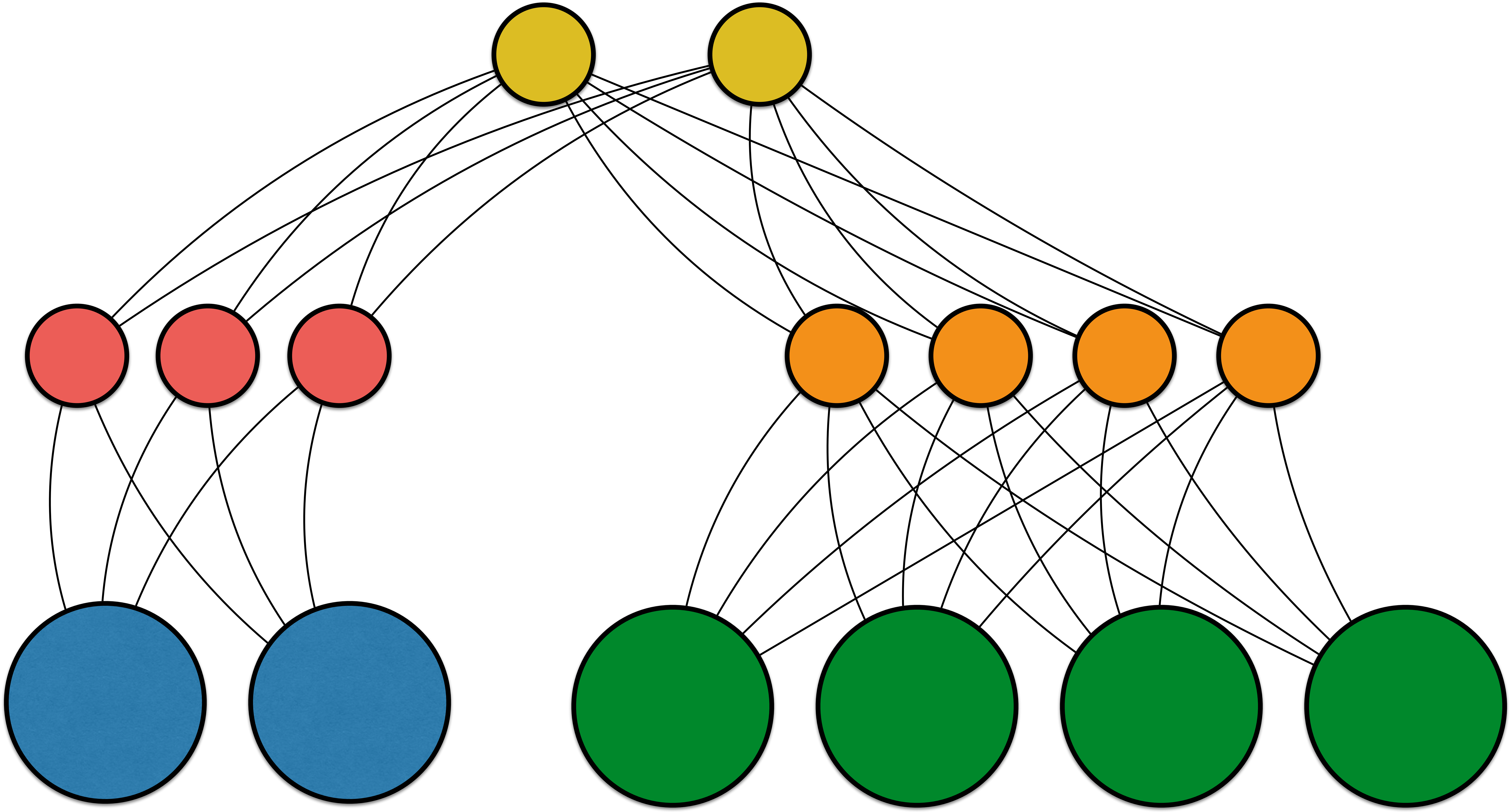
(humane)

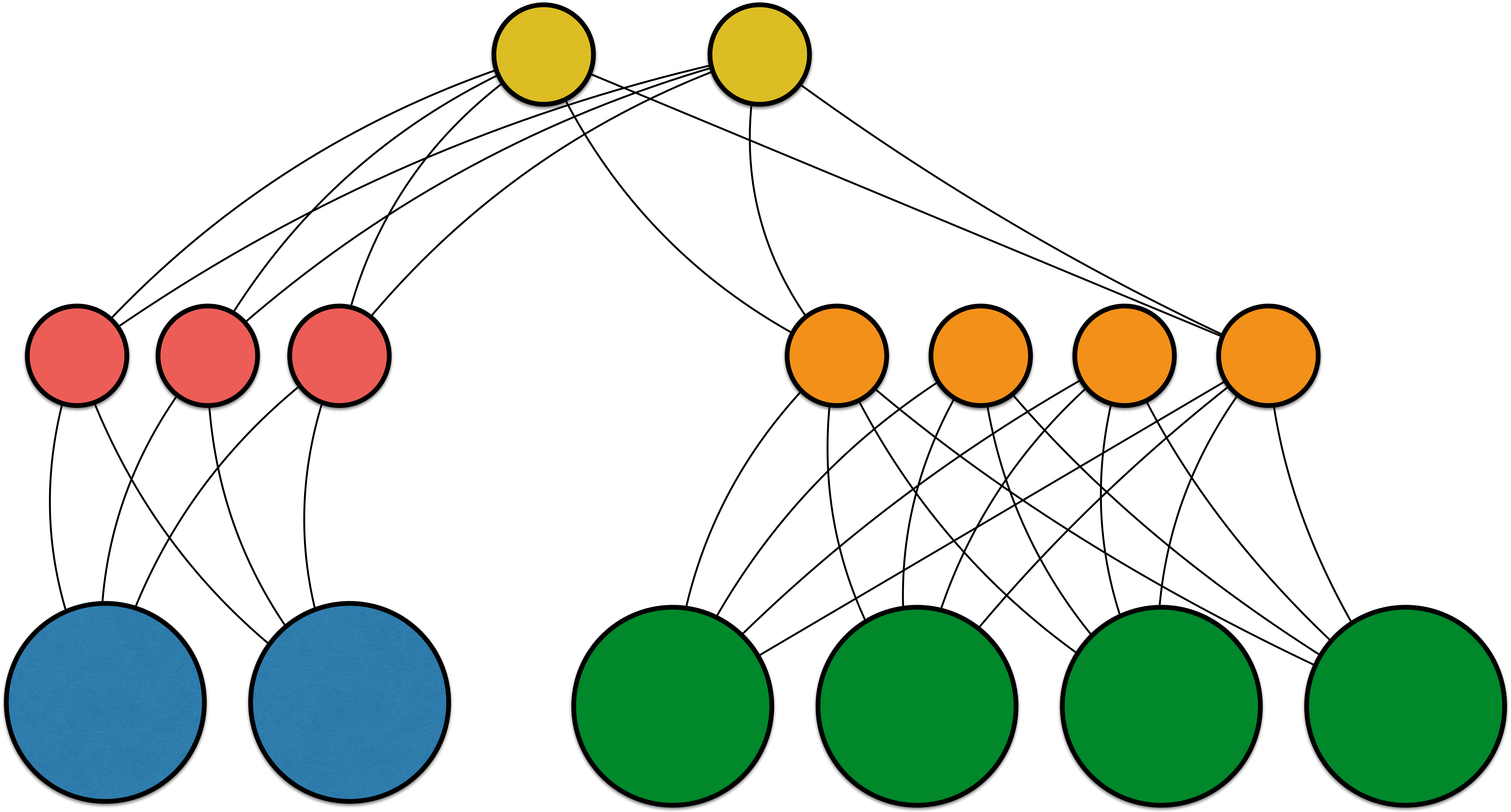


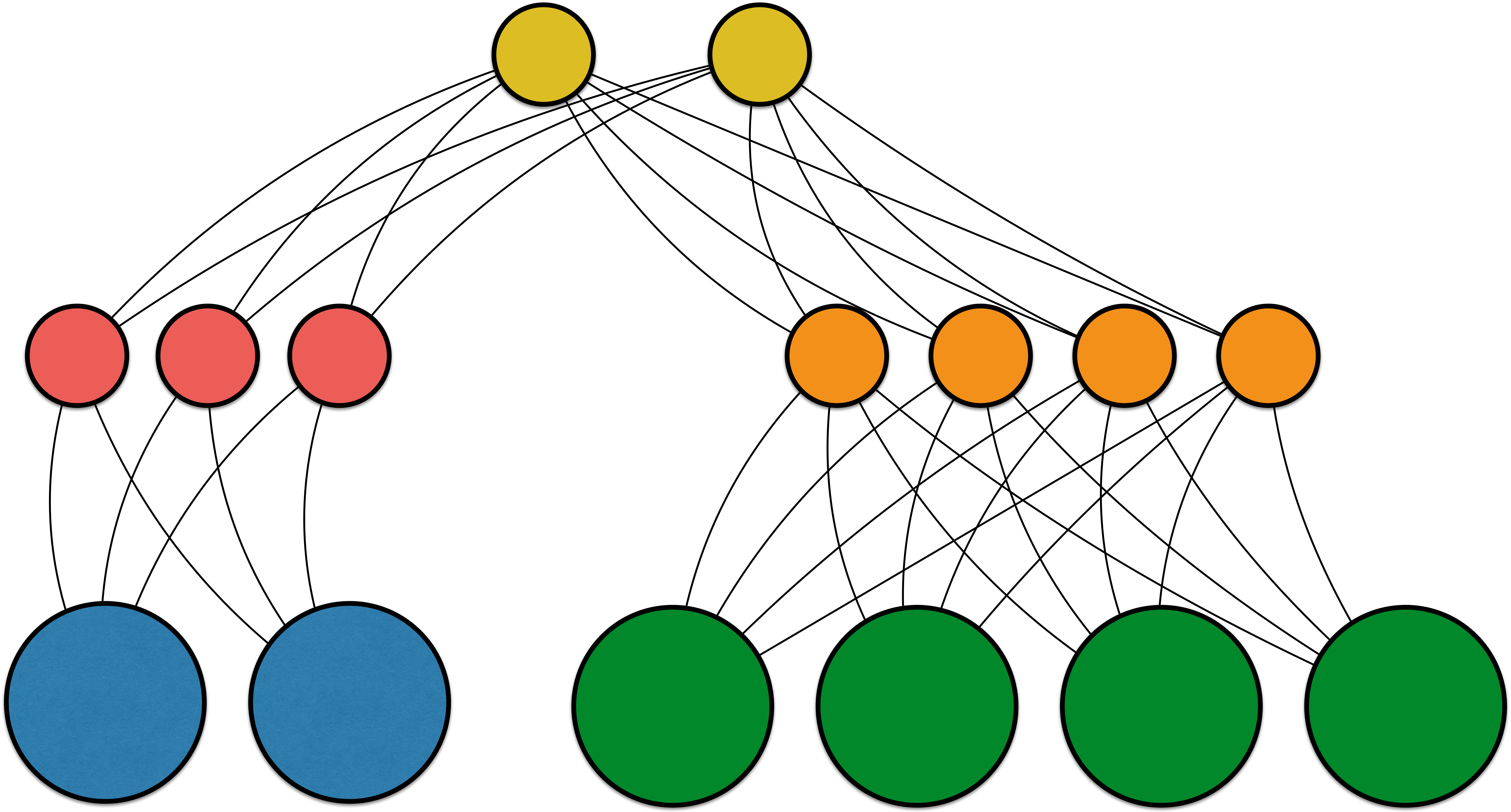
8862
paul
java
neo4j
-Xms=1024 ...
-Xmx=1024 ...
0.0.0.0:7474
95 ⇆
???
↘ 56 KB/s
10.14.0.119:30112
11 KB/s
10.19.91.119:20021
9 KB/s
↖ 997 KB/s
10.14.0.101:11323
254 KB/s
10.14.0.119:30112
331 KB/s
...
0.0.0.0:7474 →
10.14.0.119:30112
IPv4
330 KB/s
???
TCP
325 KB/s
???
HTTP
301 KB/s
HTTP 501
105/s
ICMP
45 B/s
...

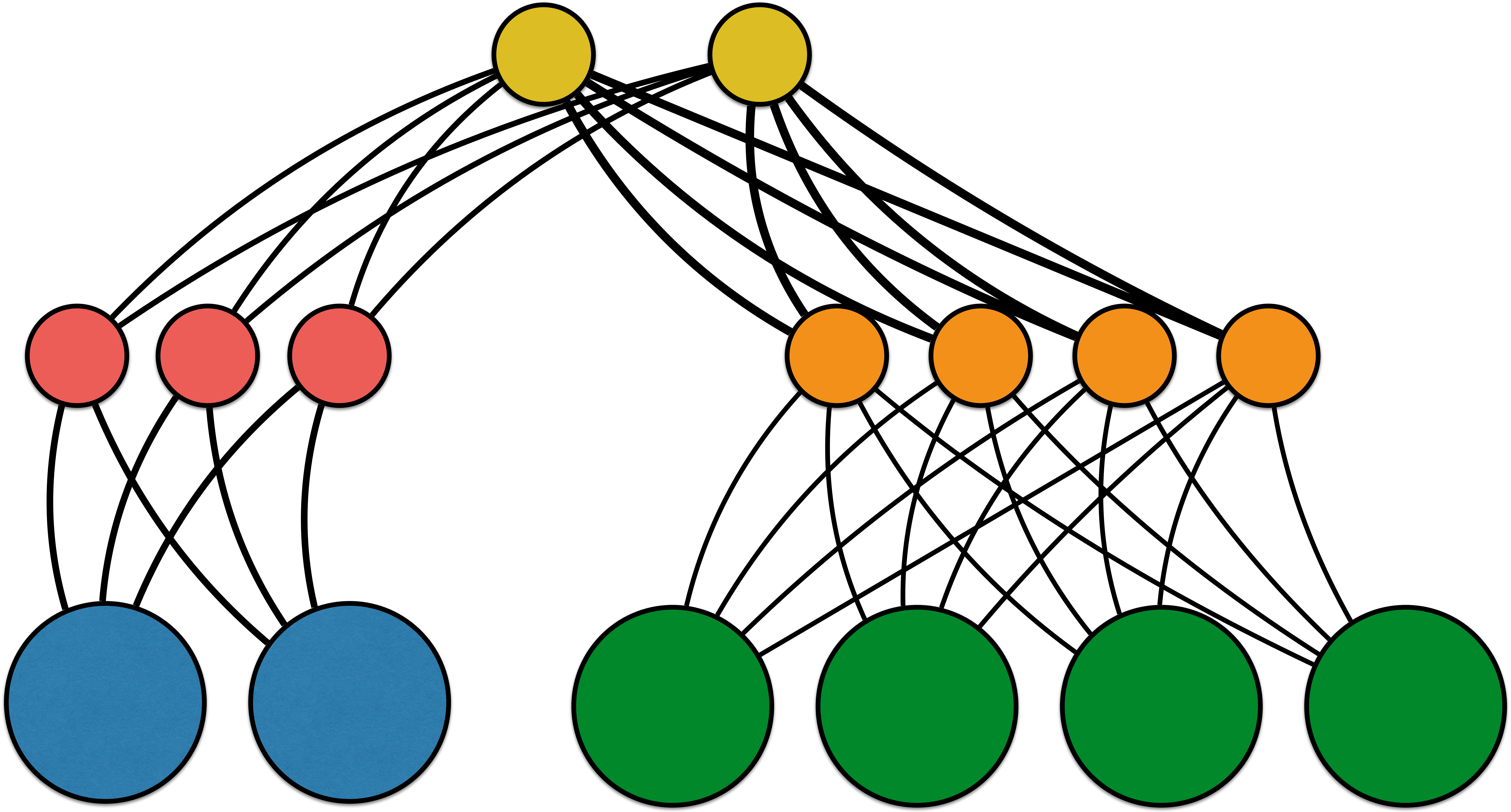


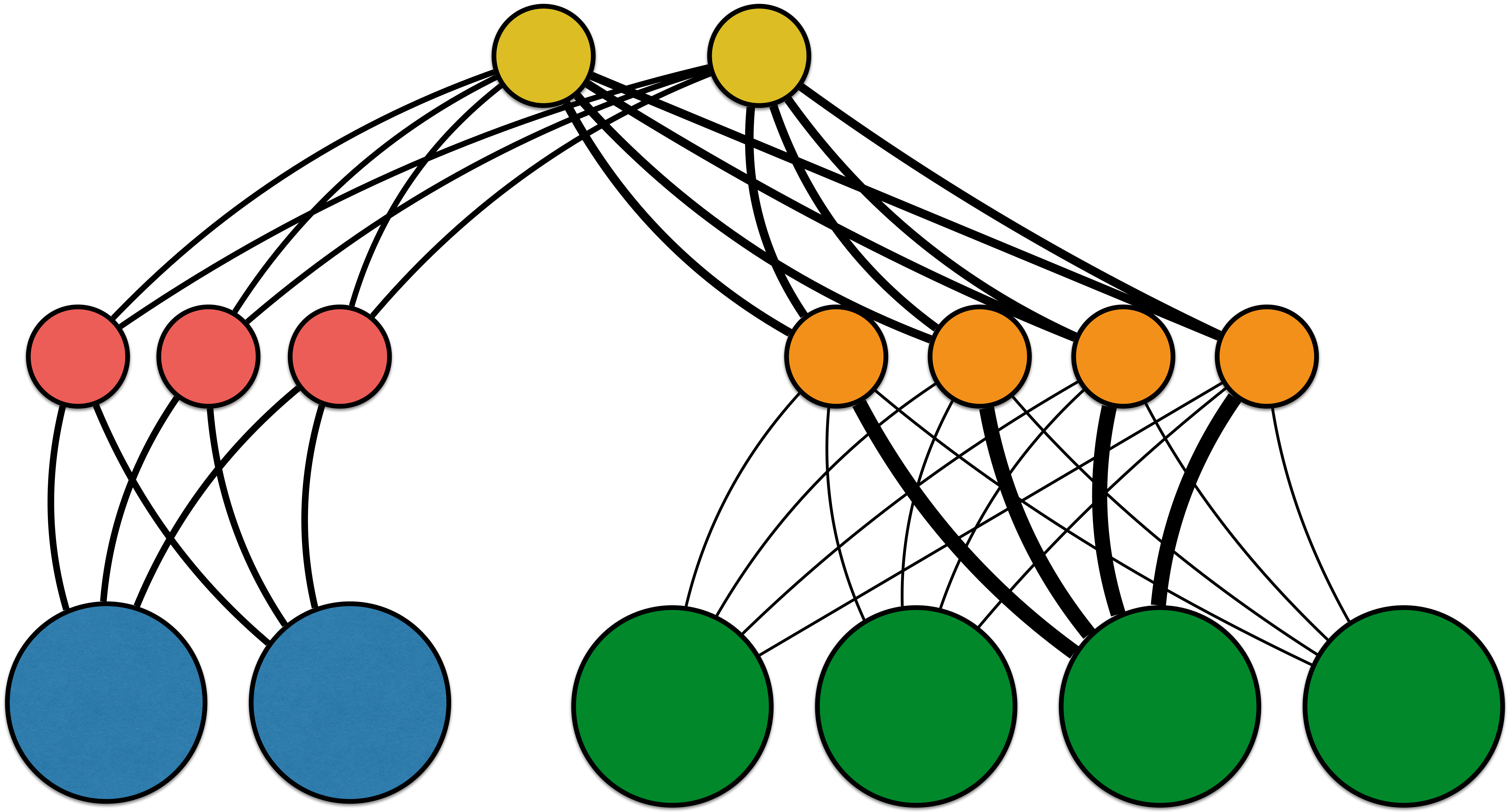


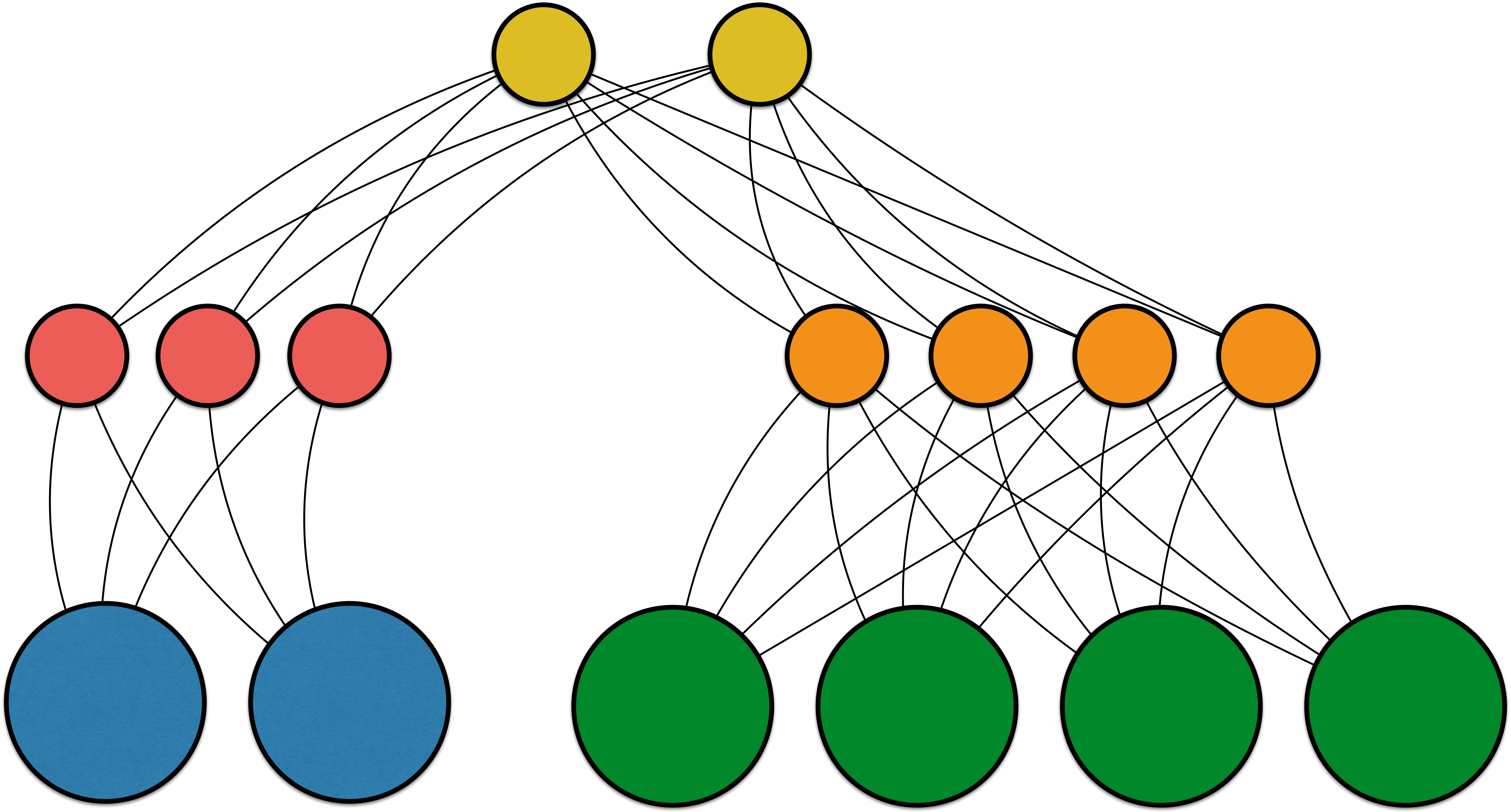


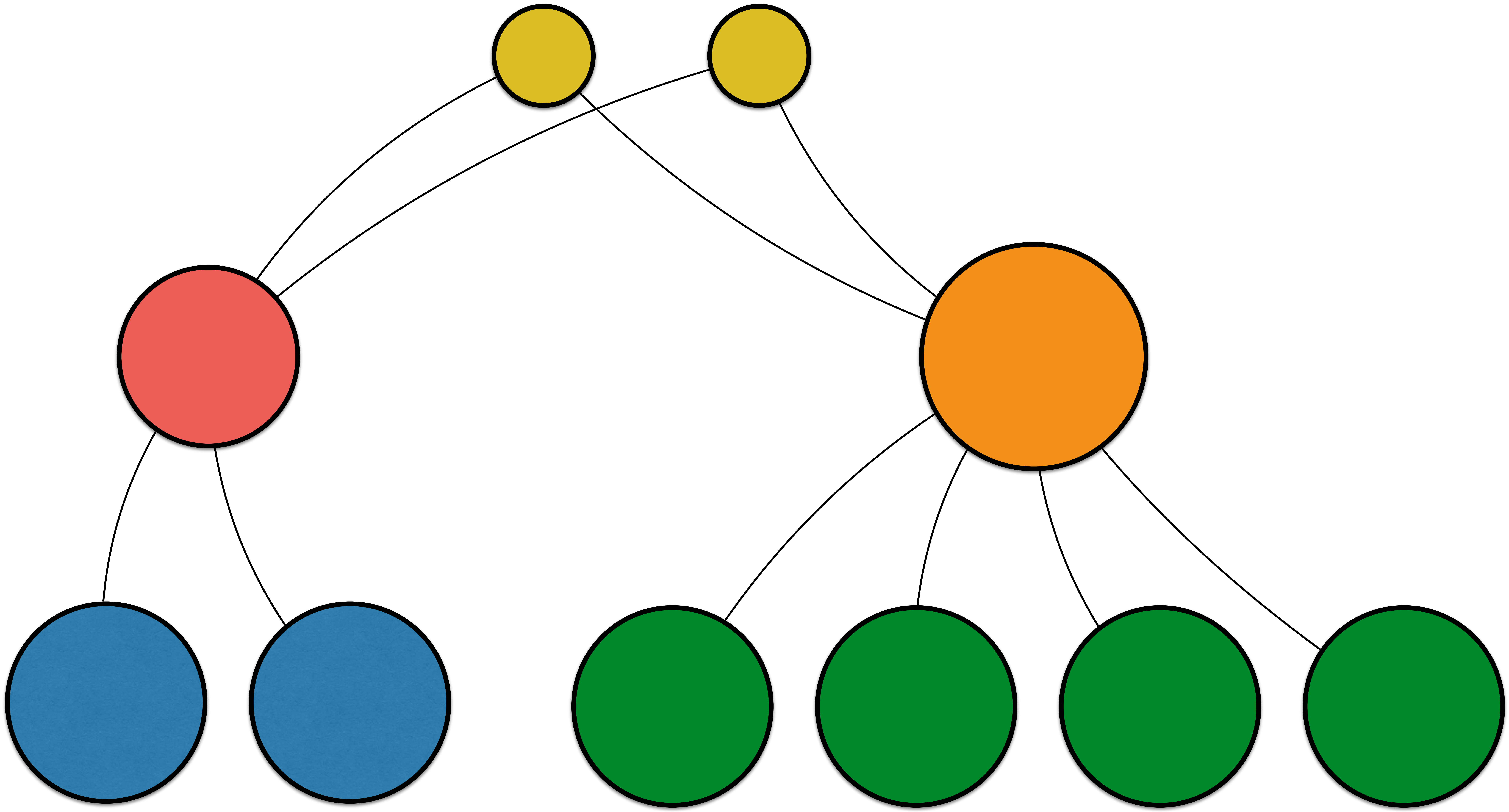


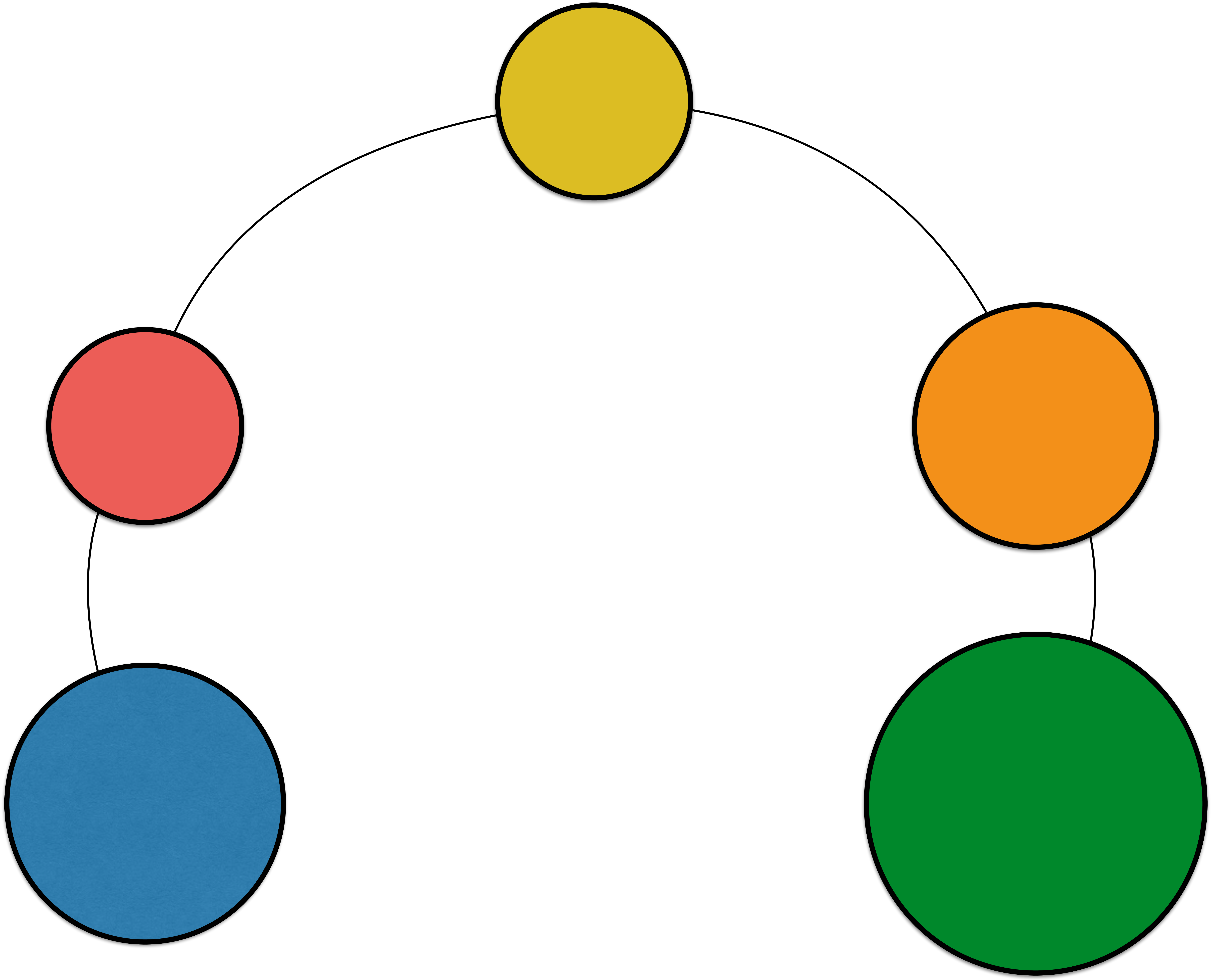












How to build it

Information sources

Invariant:

The atom of the data model is the process (PID)

Process list — `ps`

Programmatically — /proc

/proc/PID

+ complete (ish)

+ reliable

– slow (ish)

(Is there another one?)

```
pid 1234
    process_name java
    user paul
    max_cpu 101.3
    cmd java -Xmx...
    foo_bar baz
```

Communication:

Named pipes

Files on disk

Network

Communication = **sockets**

Invariant:

Communication occurs via sockets

Socket list — `netstat`, `lsof`
Programmatically — well…

Order of operations:
First: get data associated to some network ID
Later: link network ID to process ID

/proc/net/tcp[6]

· connection-based

+ fast (comparatively)

+ reliable

– just connections

tcp_diag

· connection-based

+ like /proc/net/tcp but faster

– kernel module

– not actually used?

libpcap

· packet-based

+ complete (ish)

+ can be bundled

– slow

ip_conntrack

· connection-based

+ fast

– just connections

– kernel module

netlink — nflog, netfilter

· packet-based

+ fast

+ complete (ish)

~ relatively new

– complex

Span port/port mirroring

· complete!

+ no effect on node

– separate hardware

– breaks data model :(

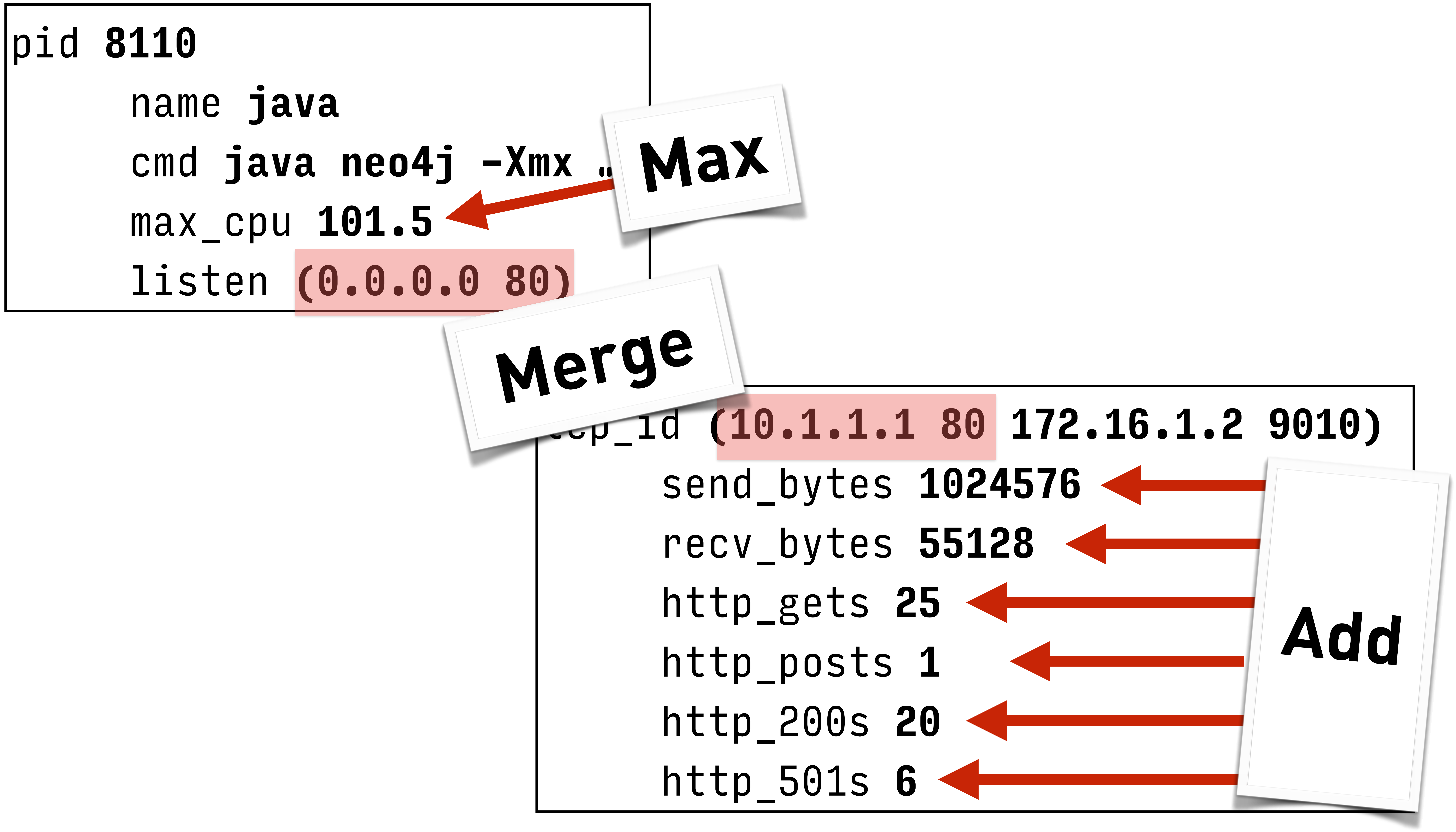
(Are there more?)

– Everything discussed is Linux
+ Other implementations possible
+ Information can compose

```
tcp (10.1.1.1 80 172.16.1.2 9010)
    send_bytes 1024576
    recv_bytes 55128
    http_gets 25
    http_posts 1
    http_200s 20
    http_501s 6
```

process ID — network ID

mapping



```
pid 8110
      name java
      cmd java neo4j -Xmx …
      max_cpu 101.5
      listen (0.0.0.0 80)
```

```
tcp_id (10.1.1.1 80 172.16.1.2 9010)
      send_bytes 1024576
      recv_bytes 55128
      http_gets 25
      http_posts 1
      http_200s 20
      http_501s 6
```

Invariant:

Observed data must merge without losing information

Data that can't be mapped
should stay in its origin domain;
∴ **multiple topologies**.

Topologies:

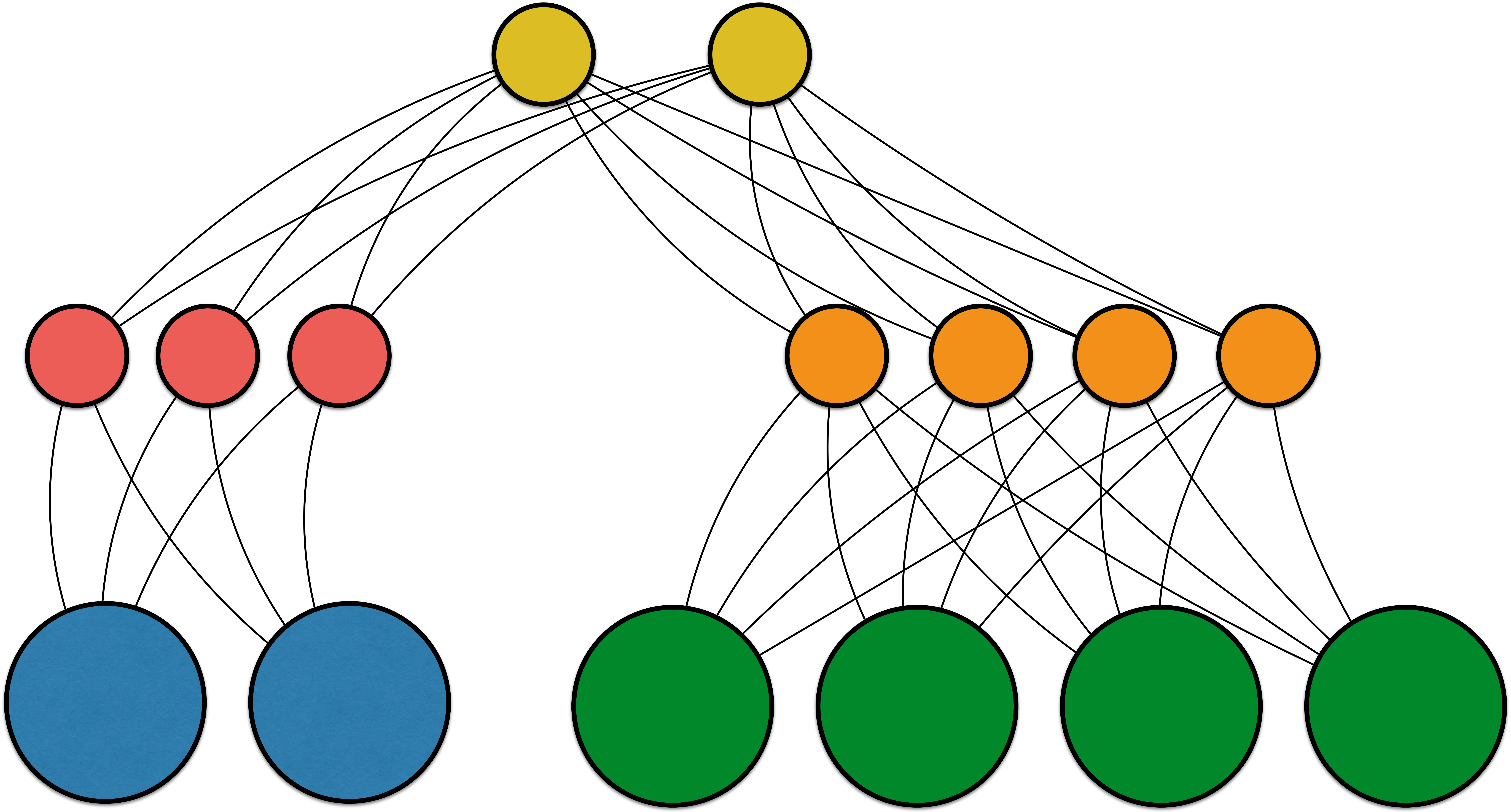
PID—PID

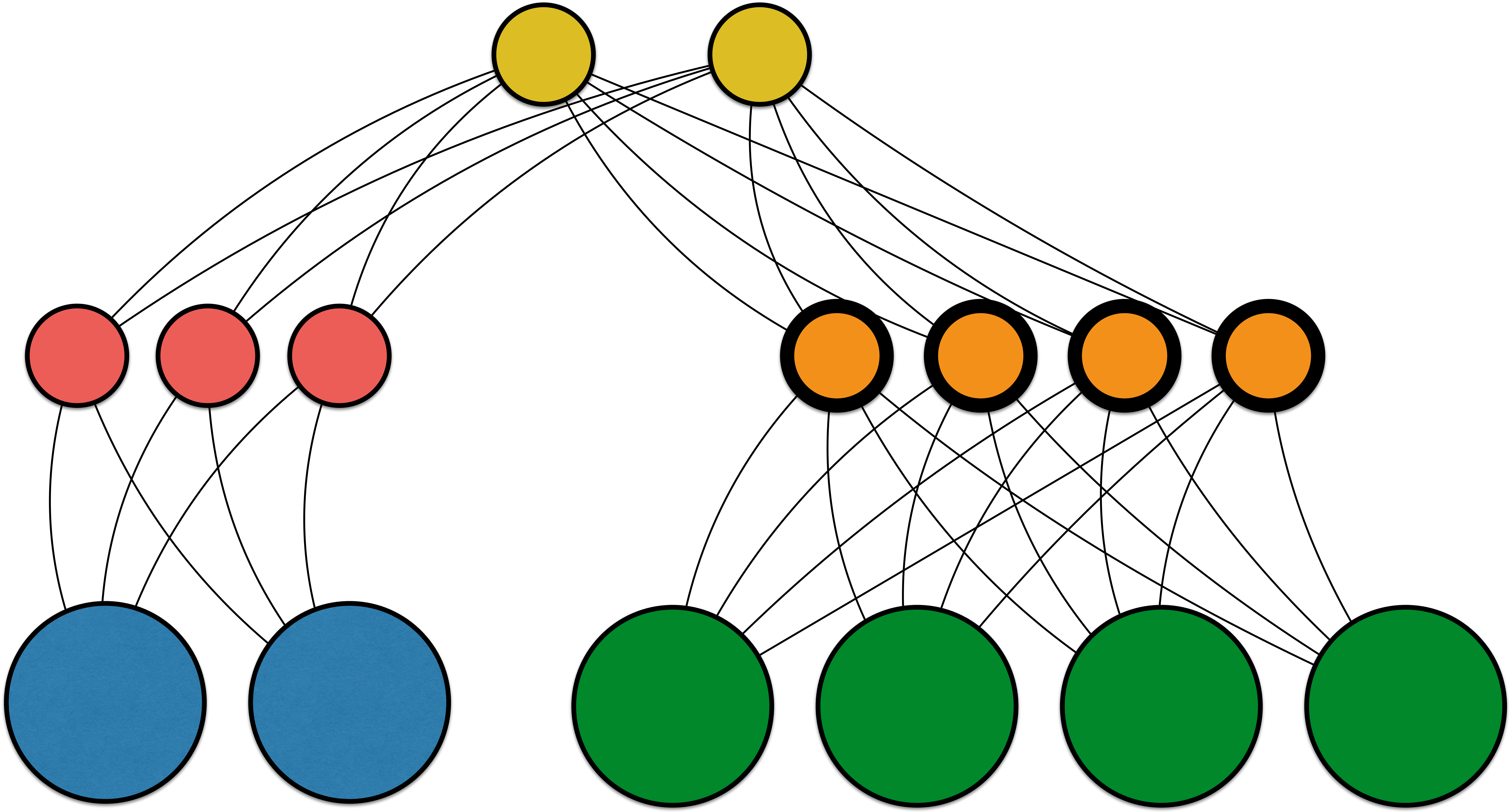
Host—Host

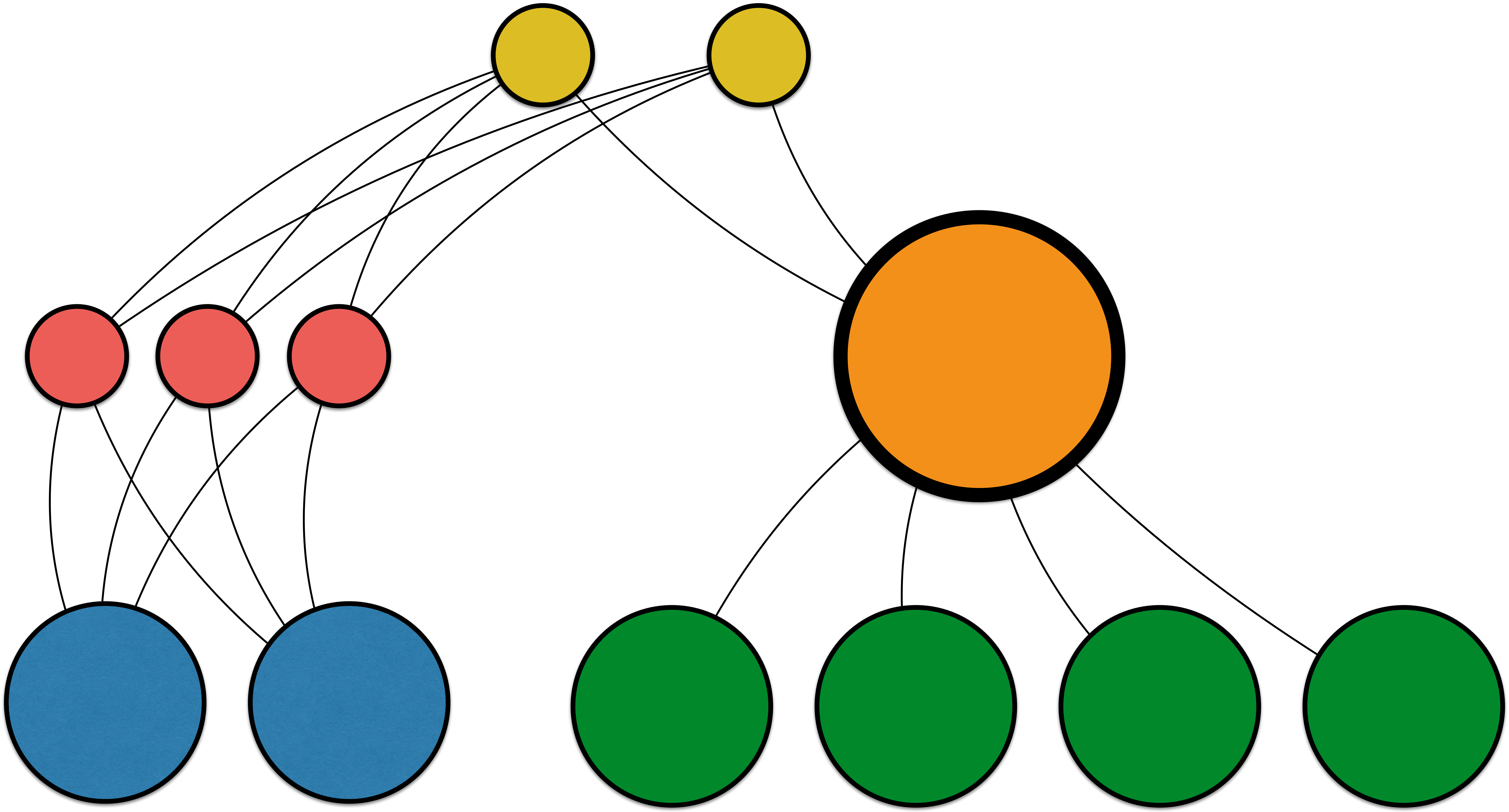
IP—IP

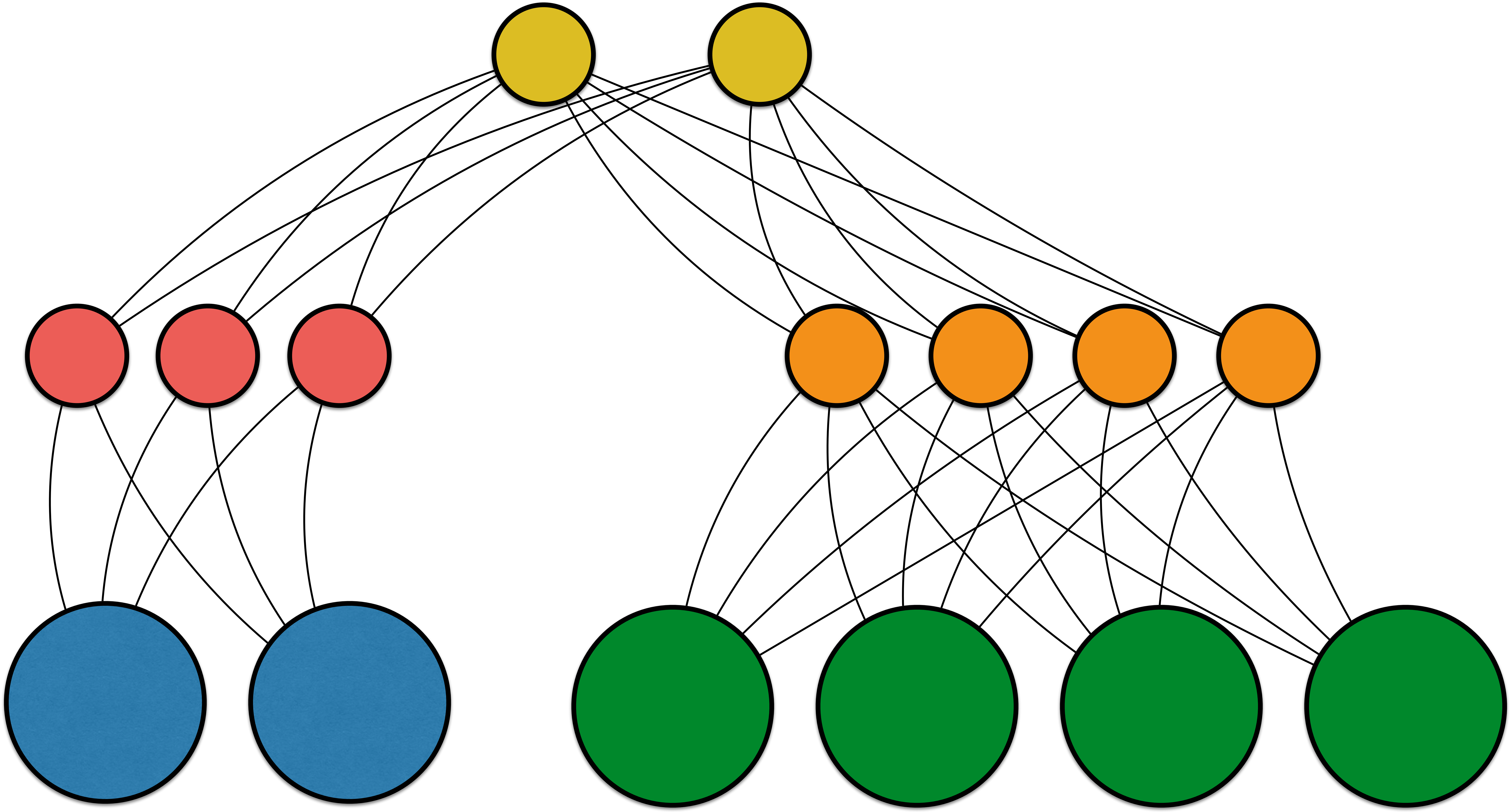
MAC—MAC?

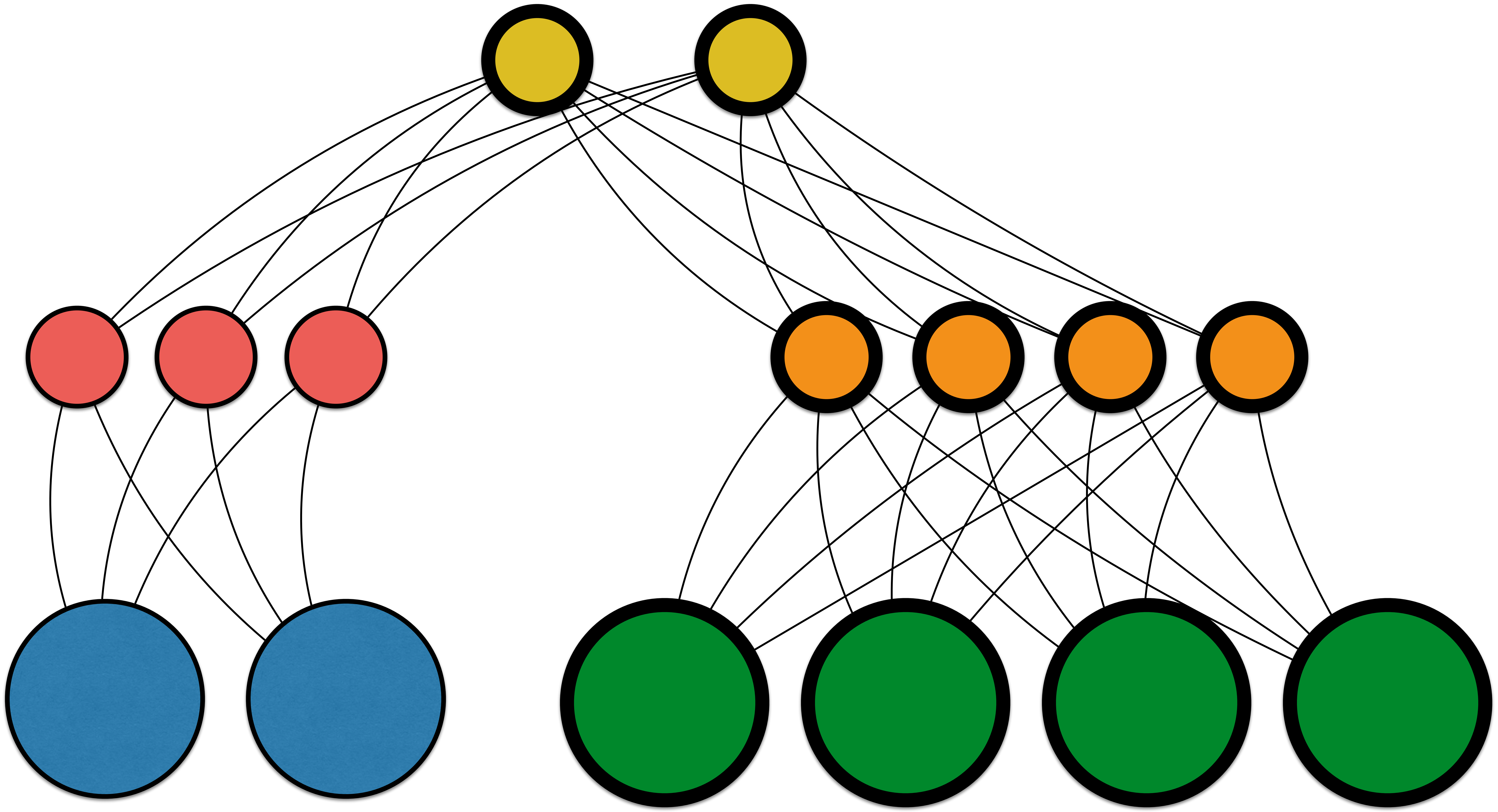
Nodes in the {**PID**, **IP**, **Host**} topology
with {**IP**, **TCP**, **HTTP**, …} traffic
{**to**, **from**, **to&from**} port/s {**N**}

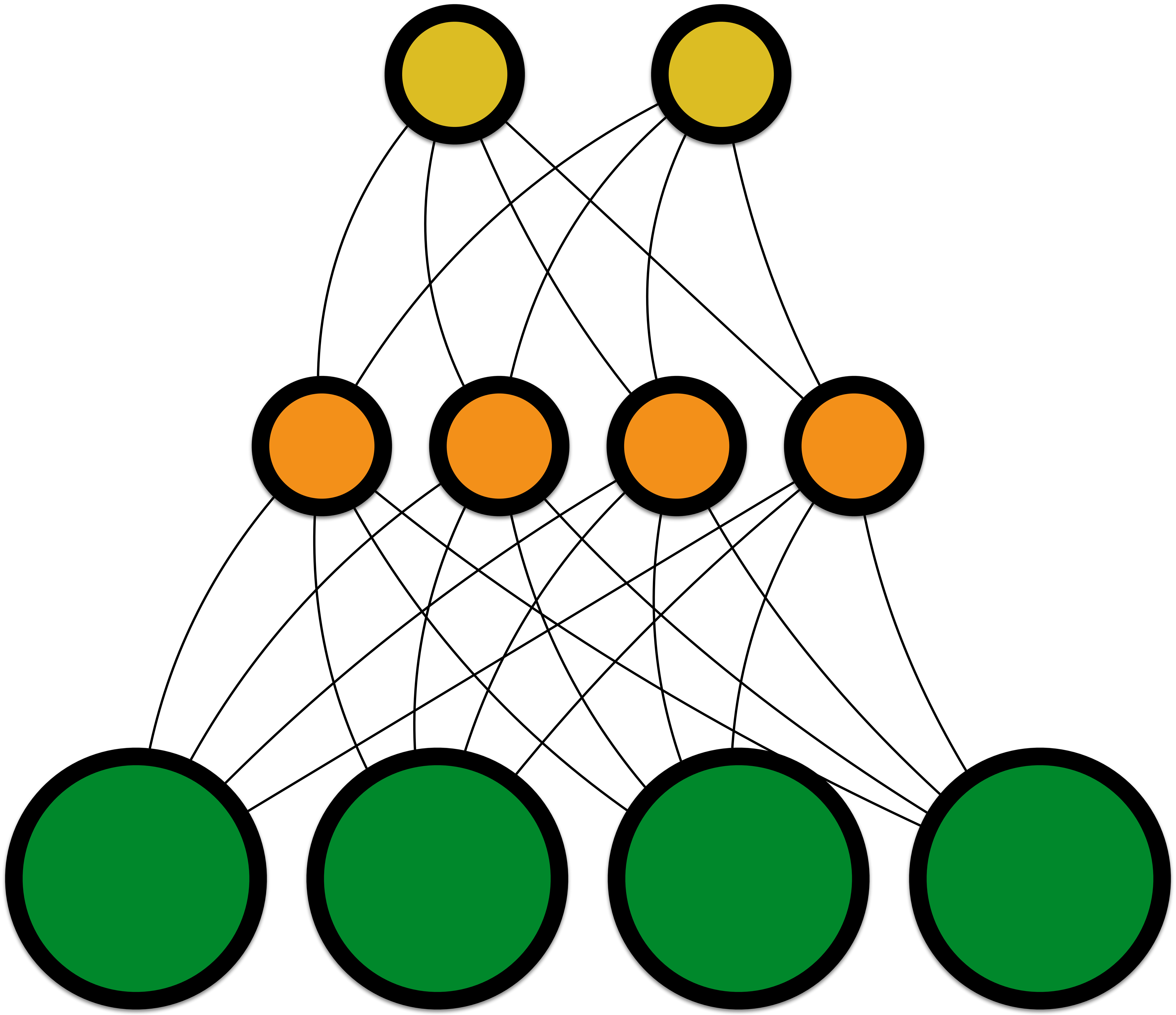






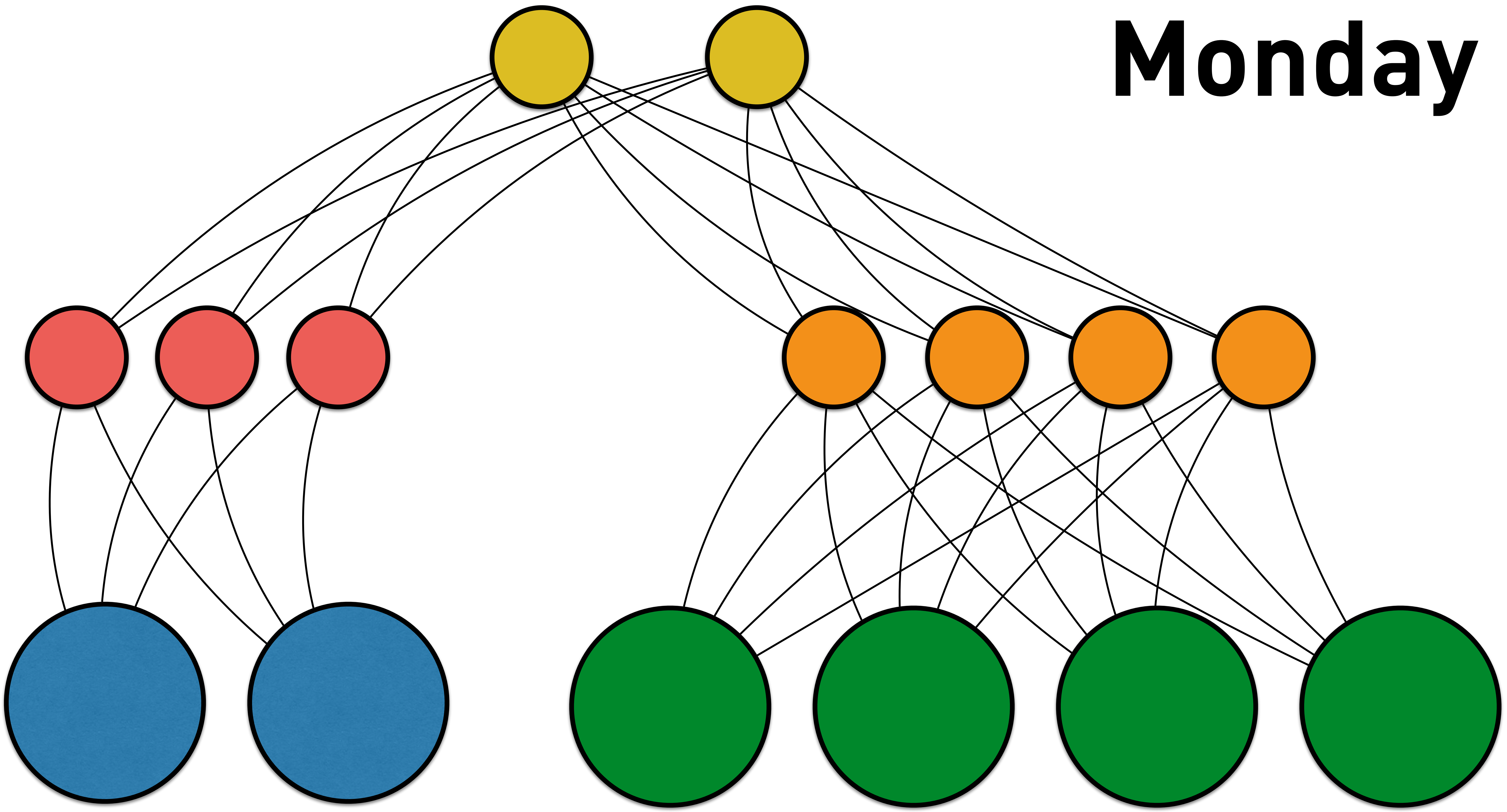


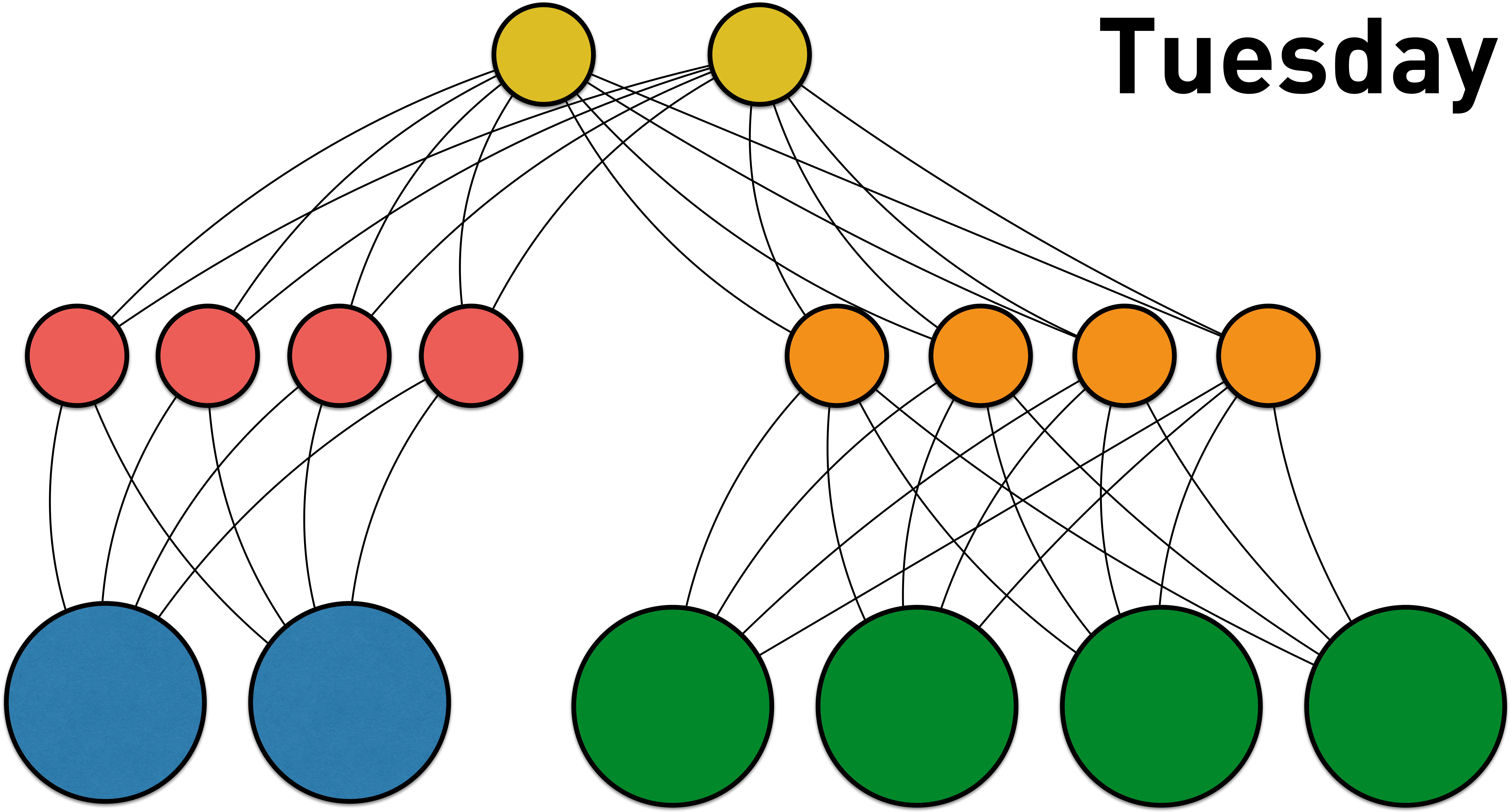




What else can we do?

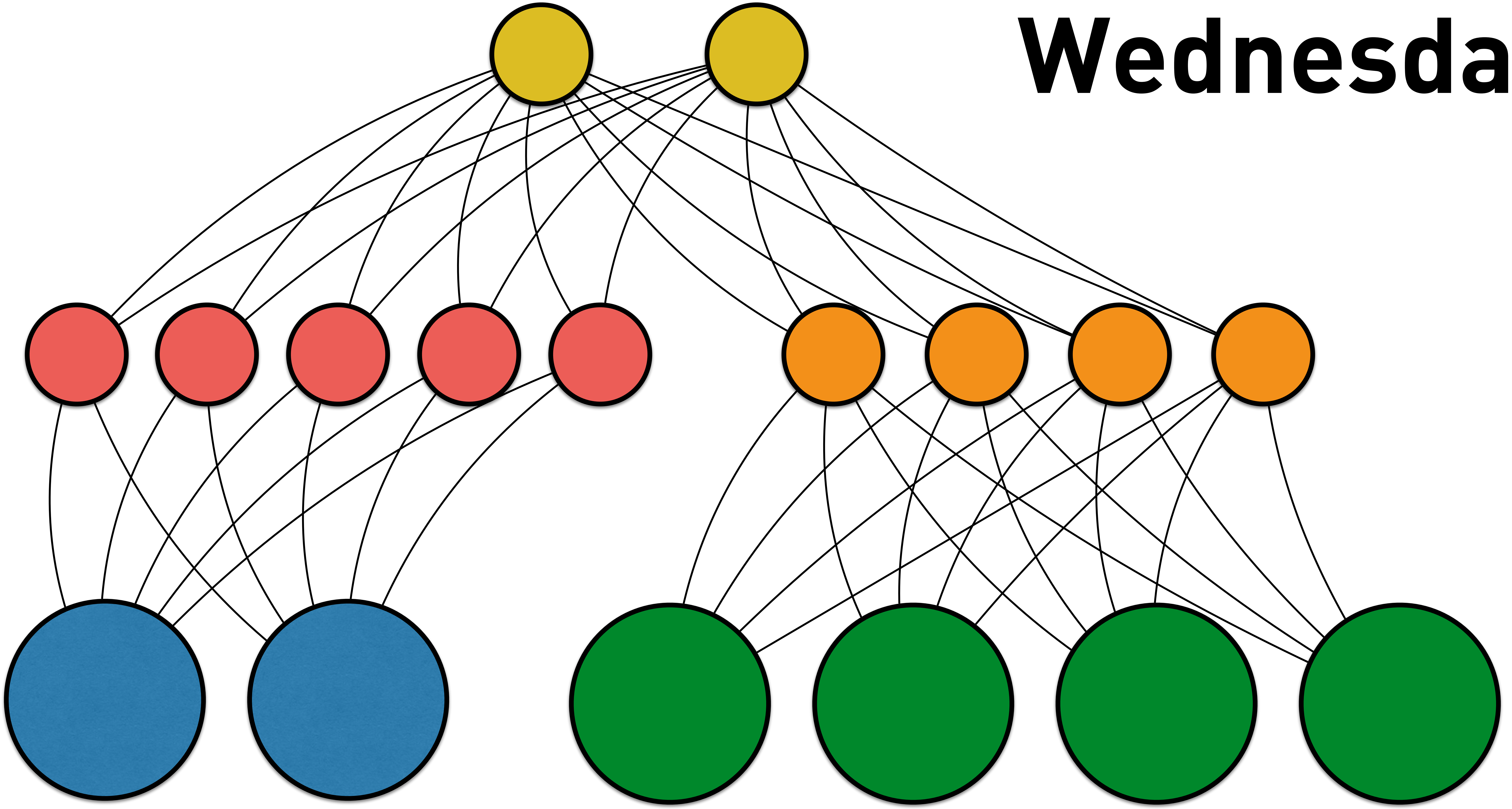
Monday

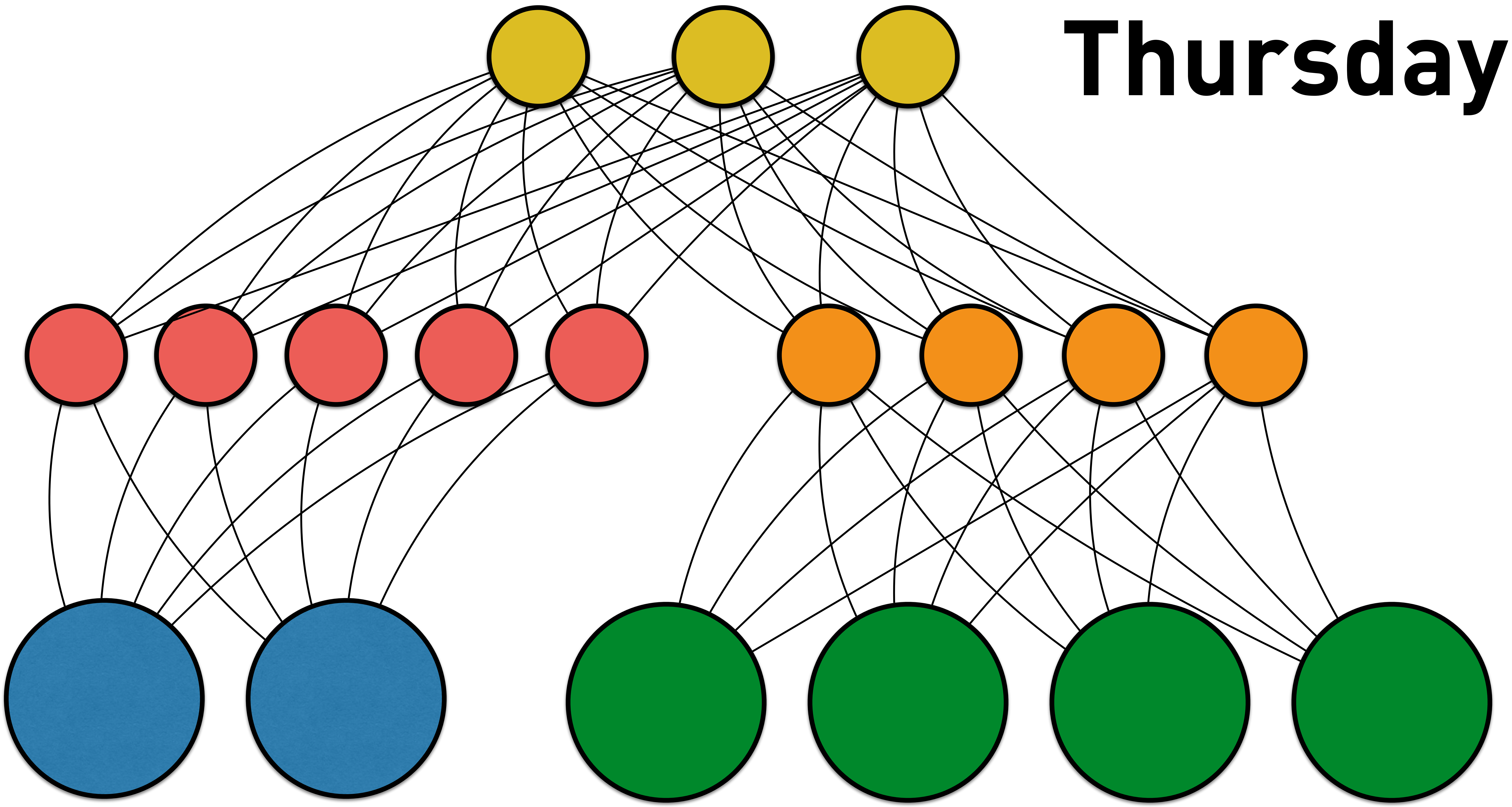




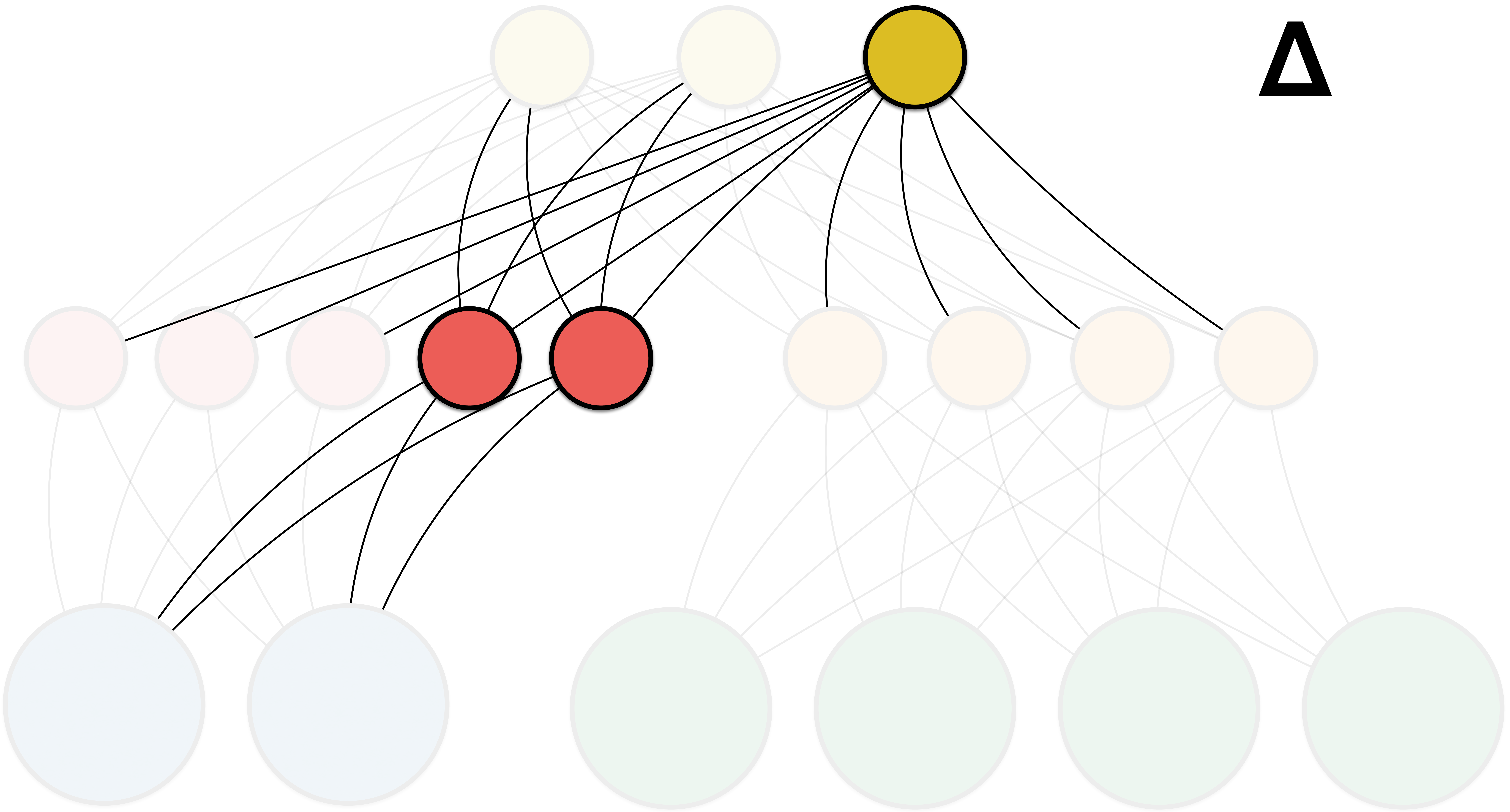
Tuesday

Wednesday





Thursday


Δ

Every data point is a time series.
Alerting, **anomaly detection**...

Conclusion

Complexity is unavoidable

Model as directed graph

An instantaneous, updating view

No configuration or declaration

Process-oriented

Communication occurs over sockets

Data must have a merge strategy

A humane tool

Focus on the facts

Help us **understand** what we’ve built

“Instead of telling me how your software will solve problems, show me … a product that is going to join my team as an **awesome team member**.”

—John Allspaw

github.com/weaveworks/scope

Thank you!

What have I missed?
What are your thoughts?

@peterbourgon