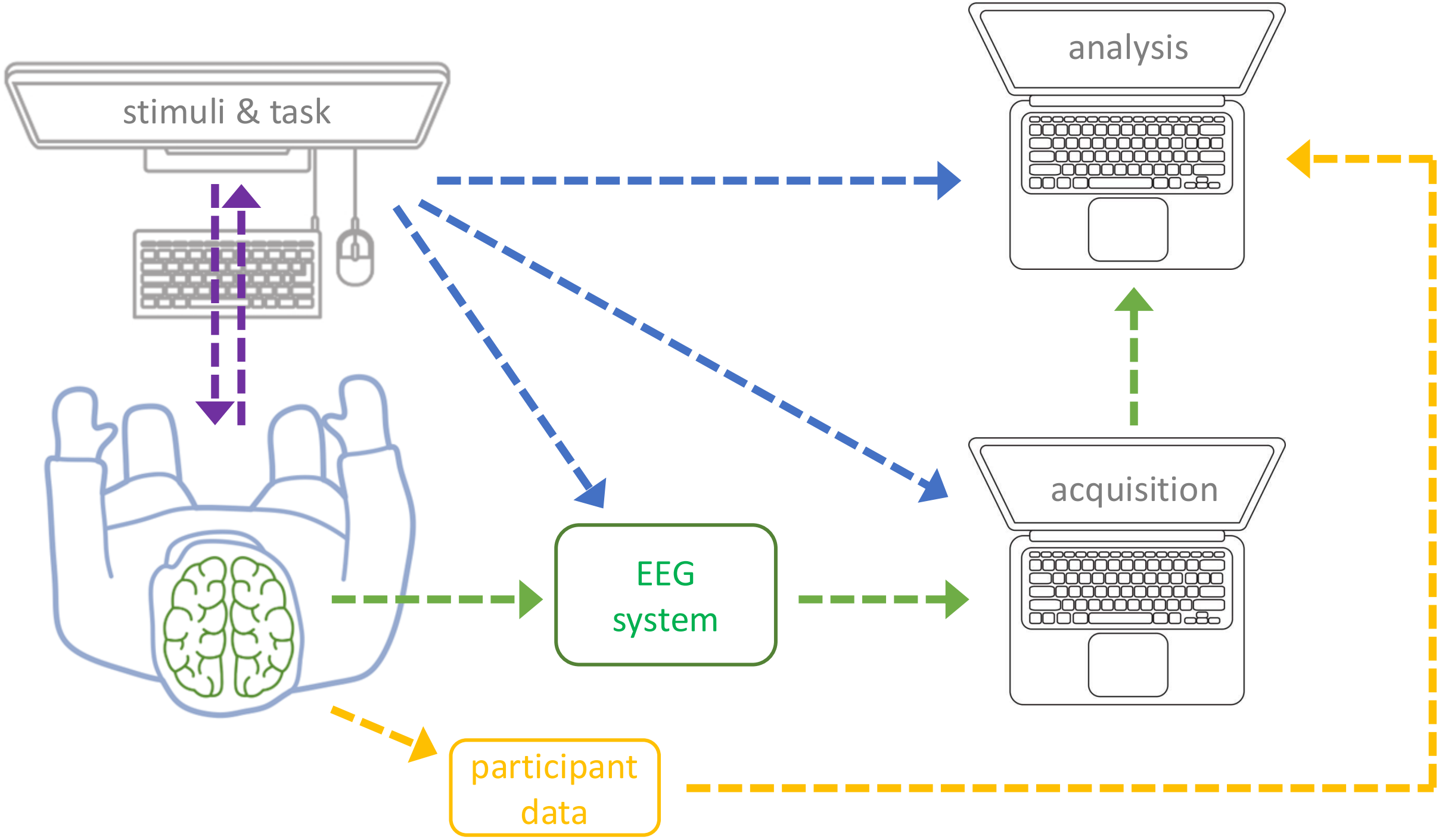
Rapid Data Analysis for Brain-Computer Interfaces

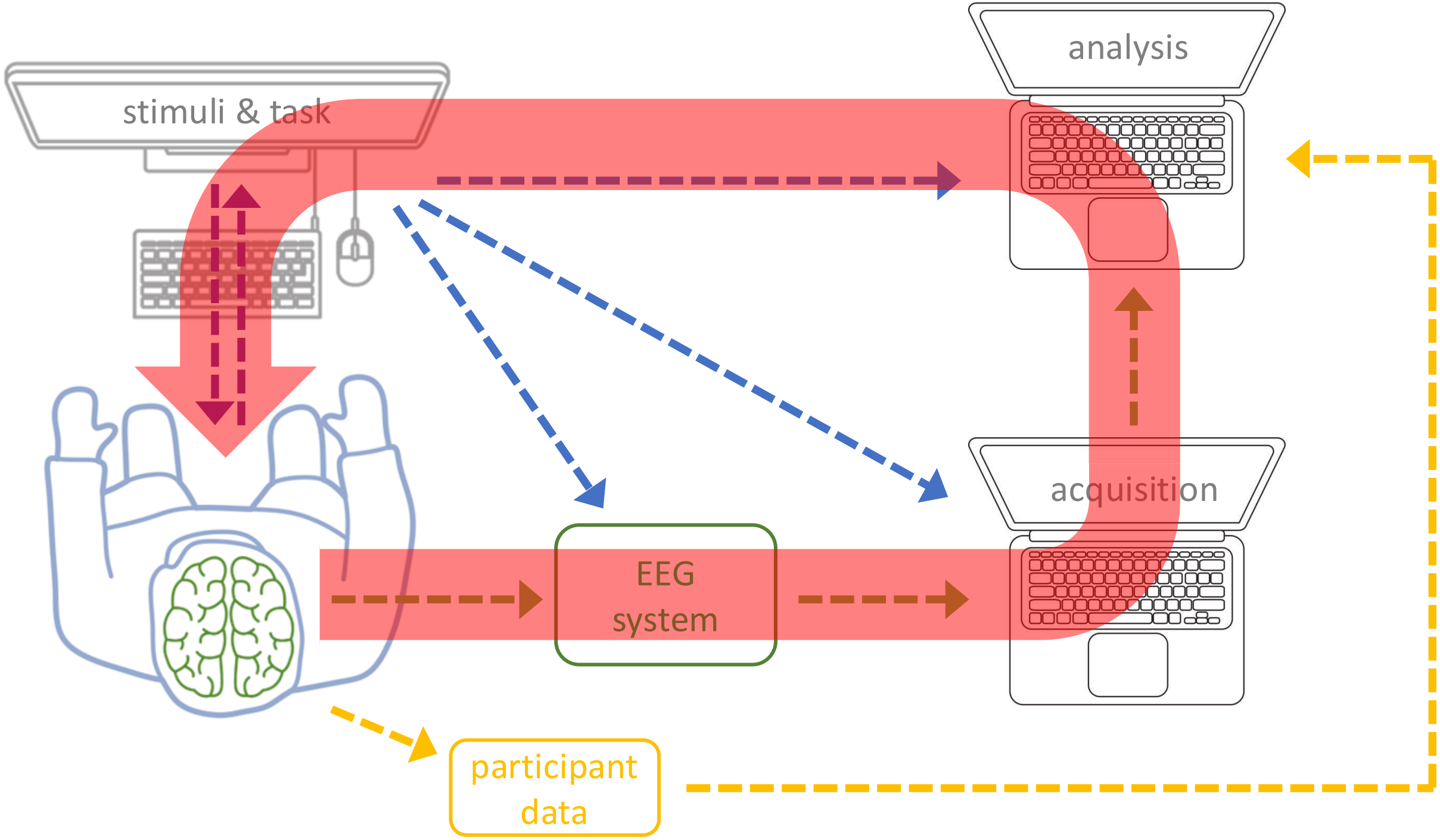
Robert Oostenveld

7 Nov 2026

NeurotechEU EEG/fNIRS hackathon

robert.oostenved@donders.ru.nl

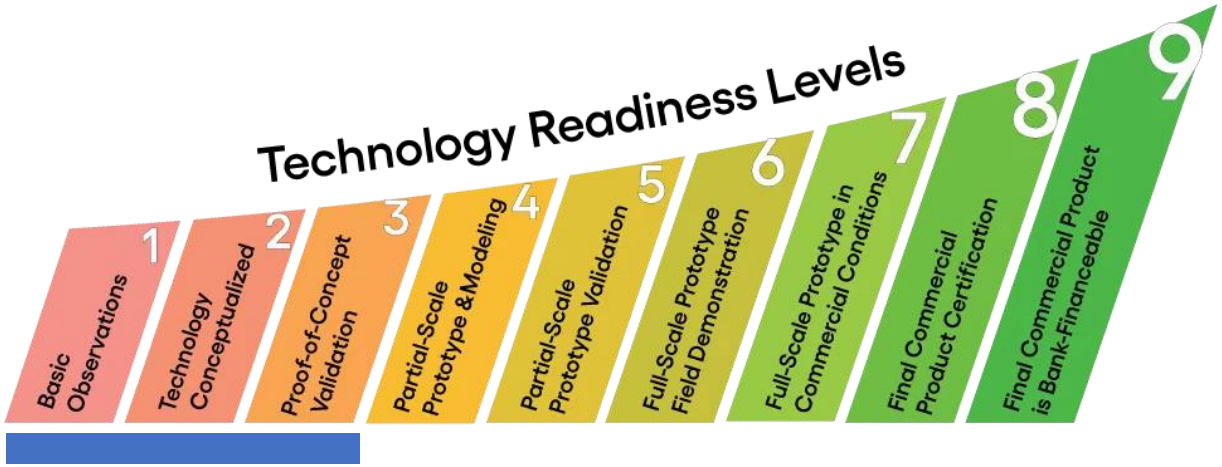

stimuli & task
analysis
acquisition
EEG
system
participant
data


analysis
stimuli & task
acquisition
EEG
system
participant
data

Brain-Computer Interfaces (BCIs)

I was not planning on presenting the general background on BCIs at this moment, as that has been covered by others in the preceding days. Any questions before I proceed?

Offline analysis



Proof of principle of the neuroscience

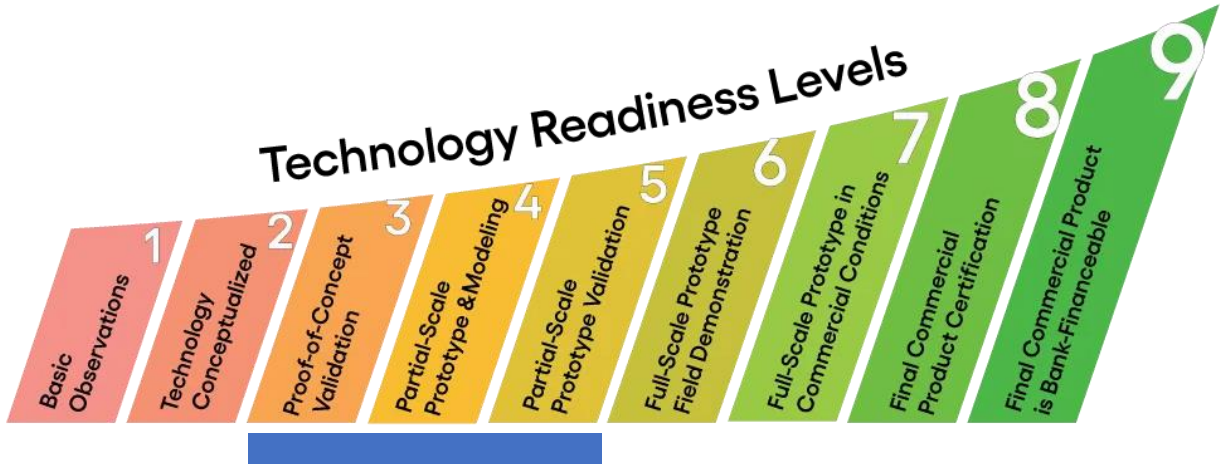
Learn about the required experimental manipulations

Use data from many participants, many trials

See whether data processing and feature detection generalizes

Feasibility of data acquisition and quality

Realtime online analysis



Translate group study into single-subject and single-trial analysis

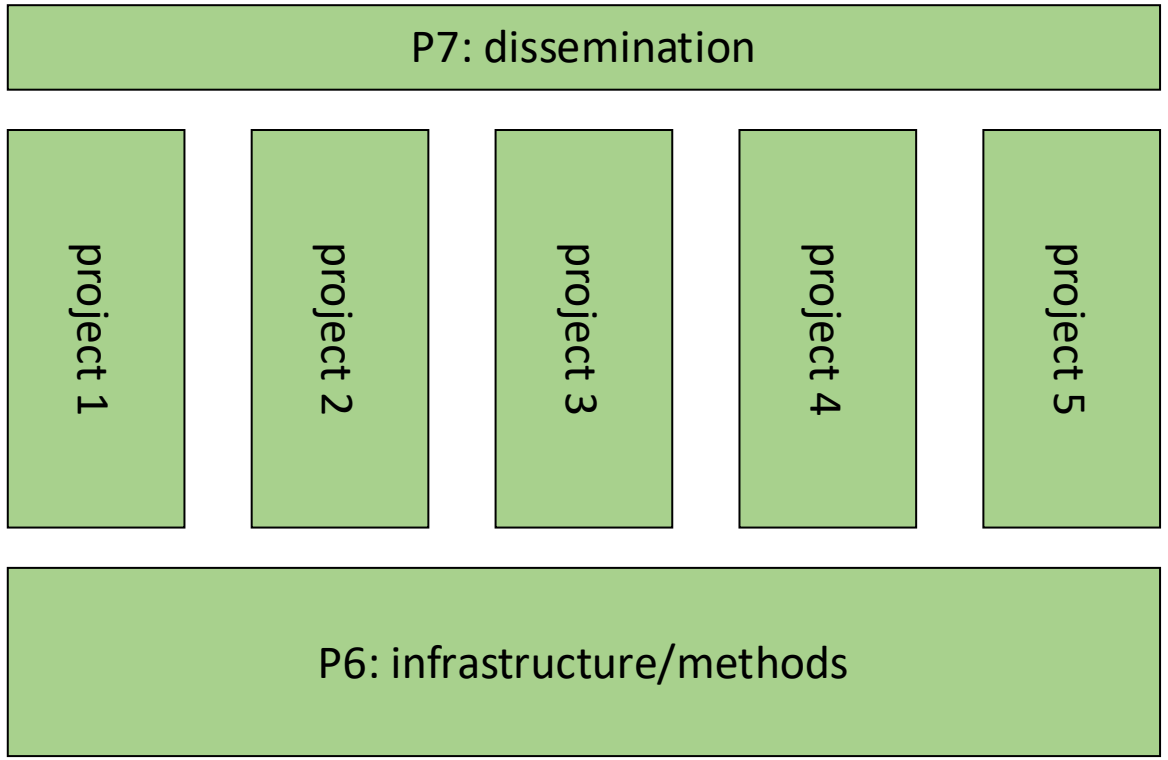
Change the experimental setup to real-time

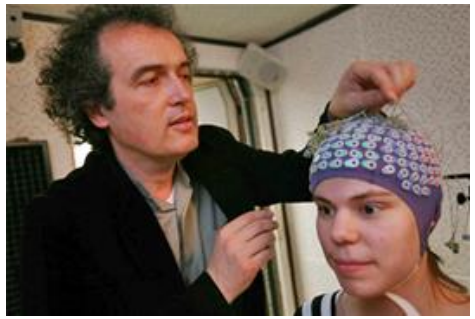
Signal detection and classification

Provide brain-based feedback or control

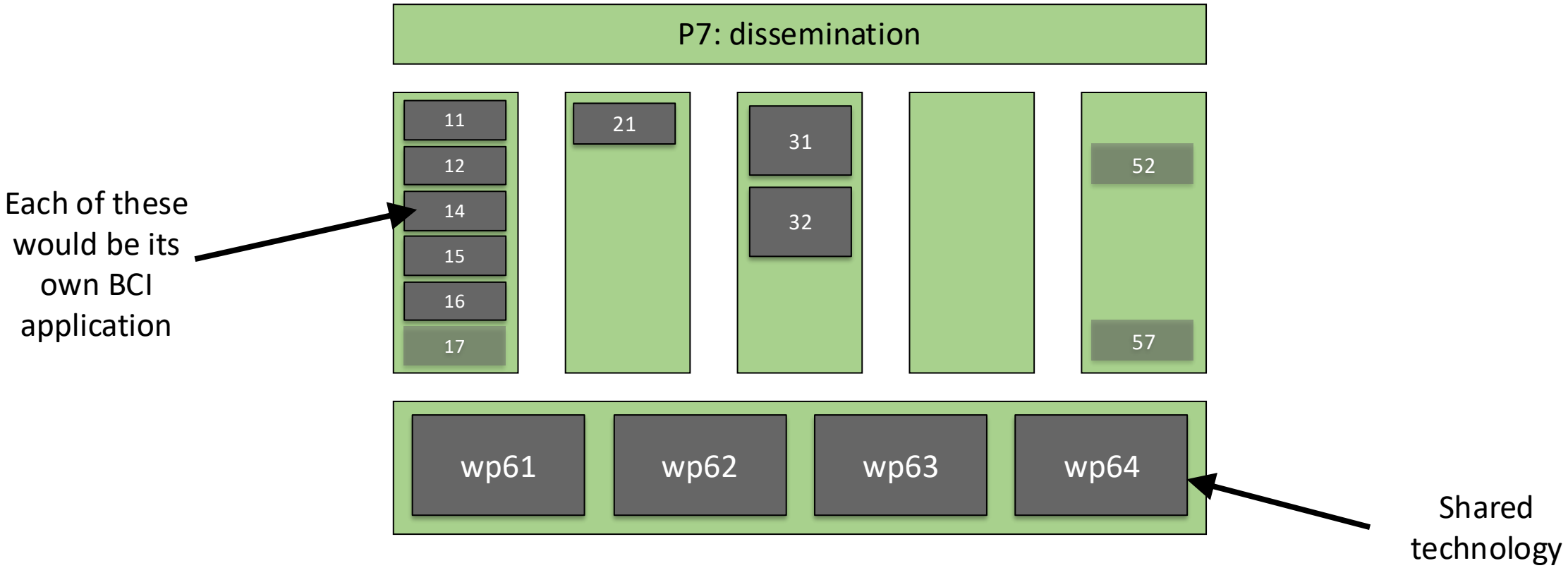
Some history from the BrainGain project - overall structure

BrainGain was a 5 year (2007-2012) government-funded Dutch BCI project with many university and industrial partners, led by Peter Desain at Radboud University.

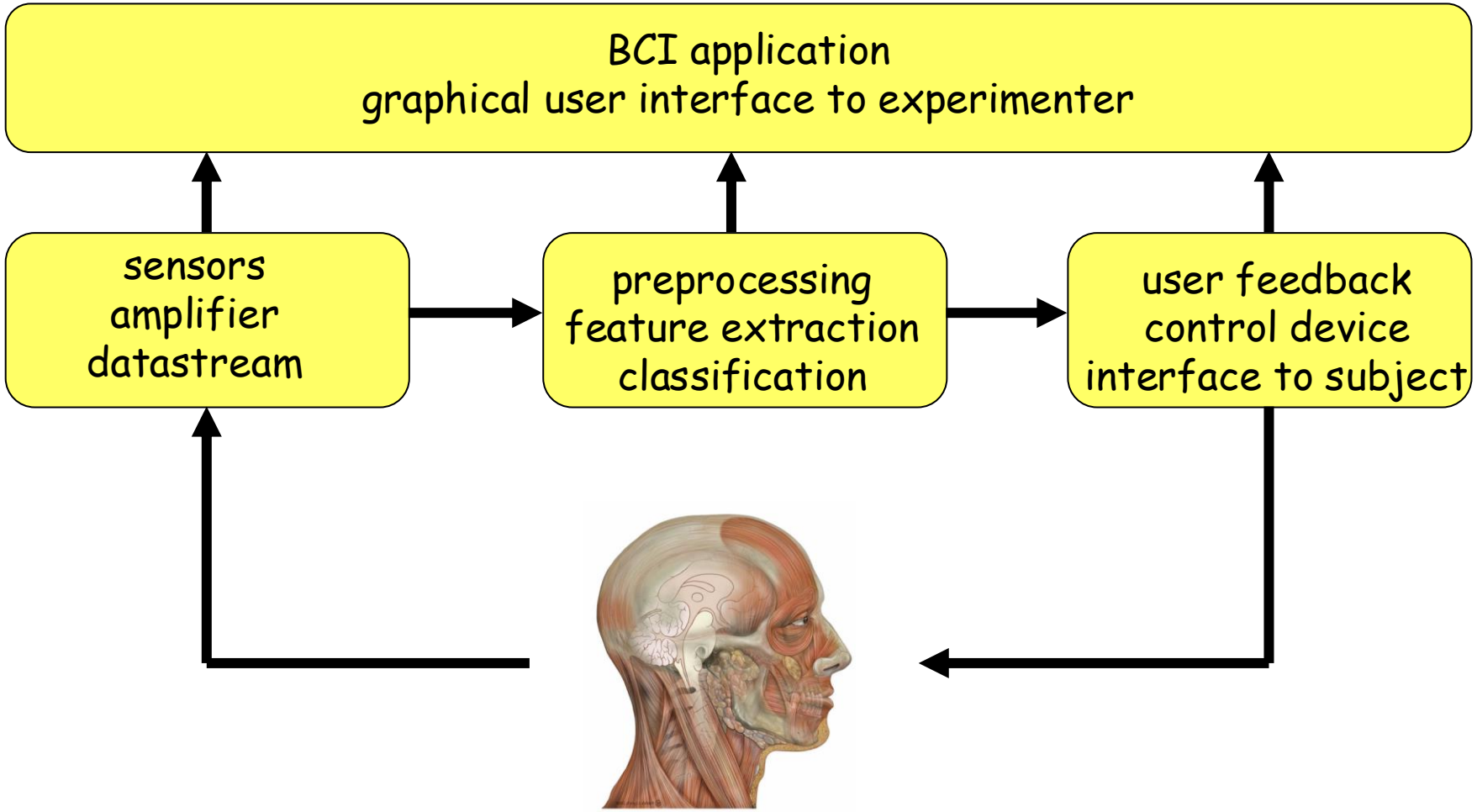




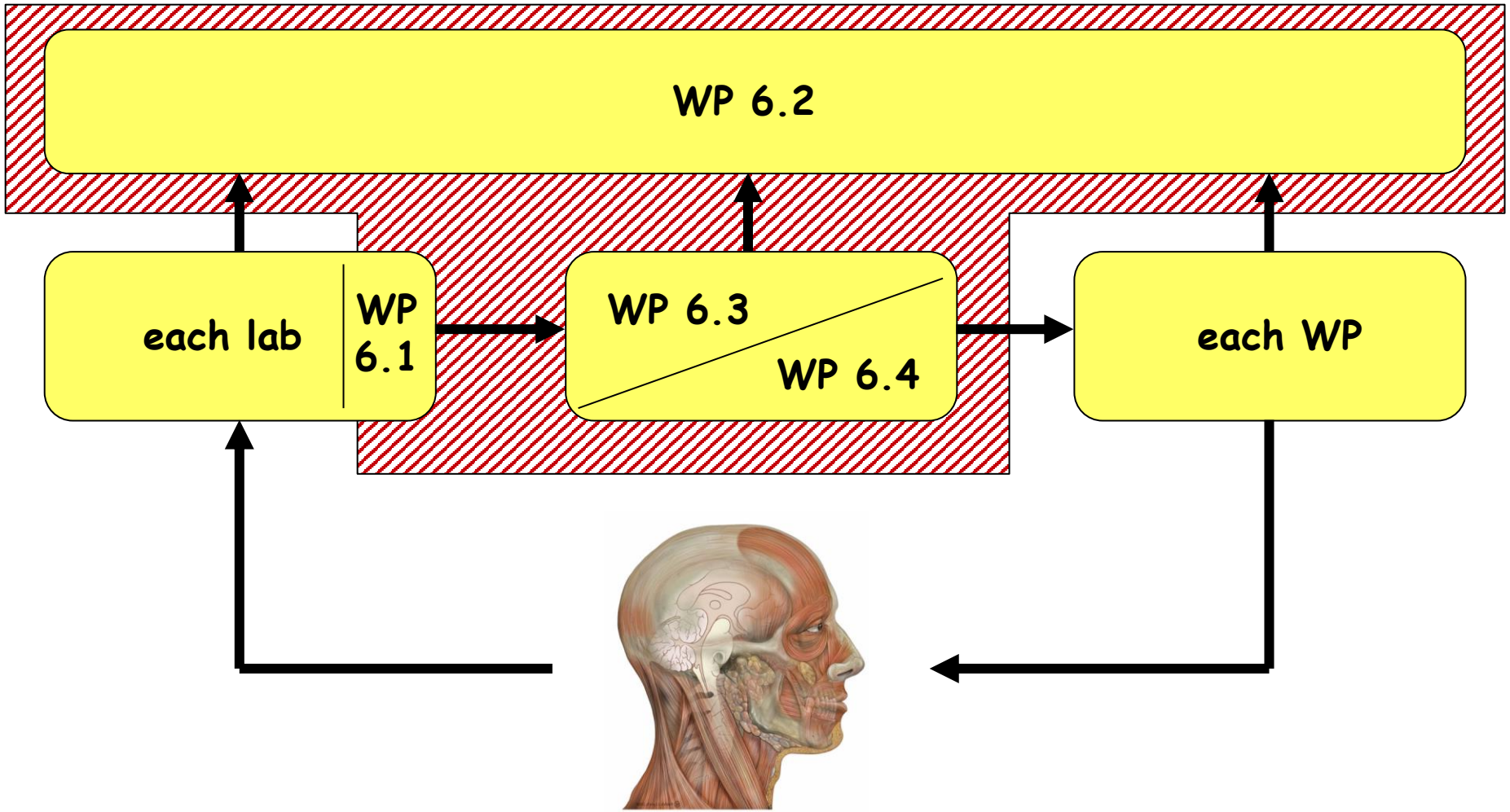
BrainGain projects - split up in workpackages



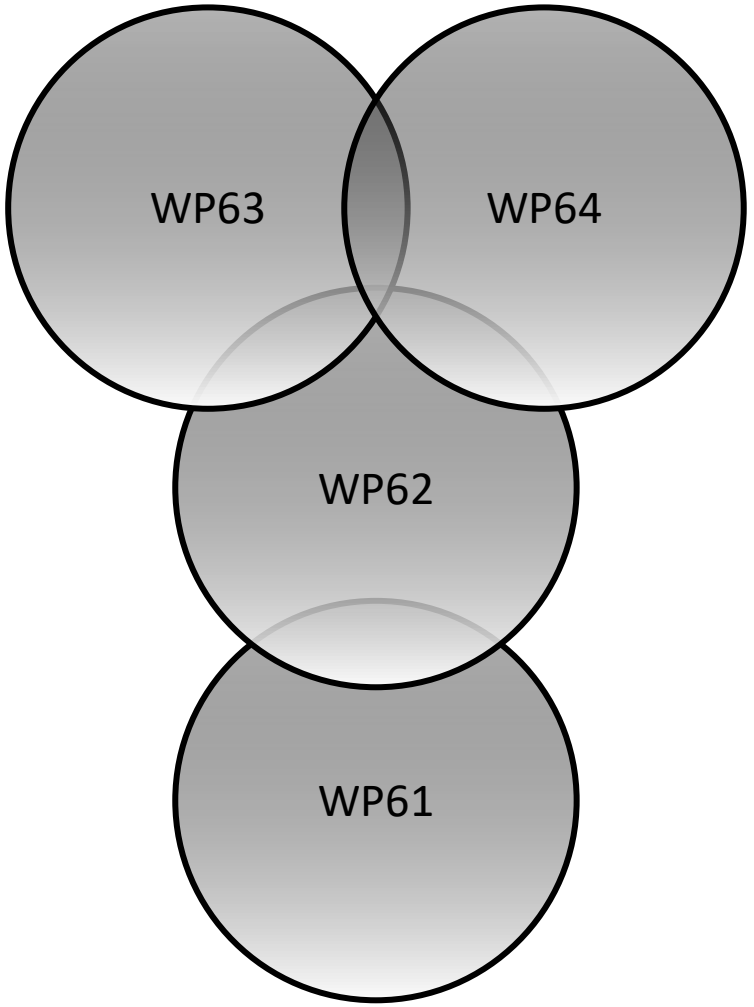
Closed-loop BCI according to BrainGain, BCI2000, OpenVibe, TOBI, …



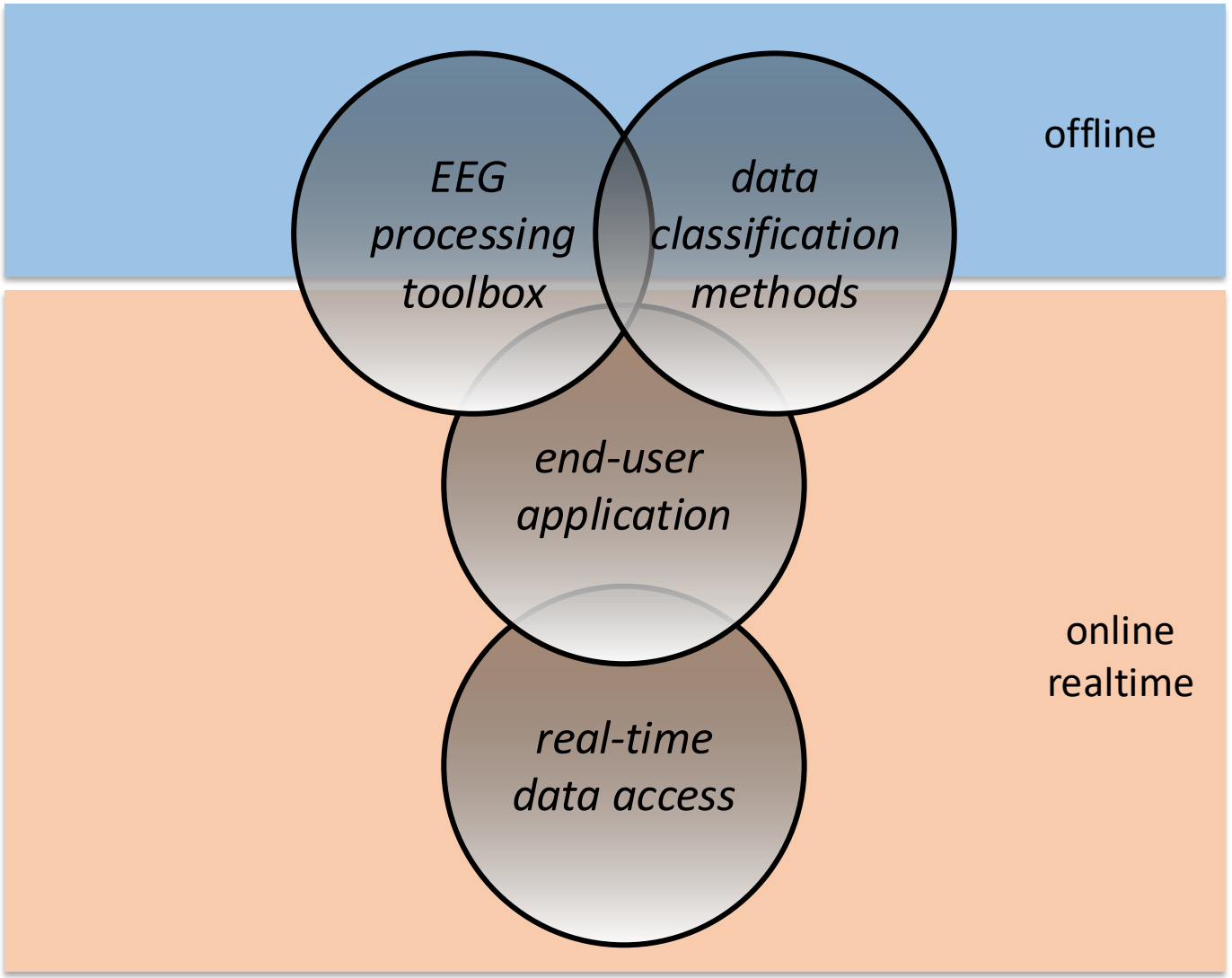
Scope of BrainGain project 6



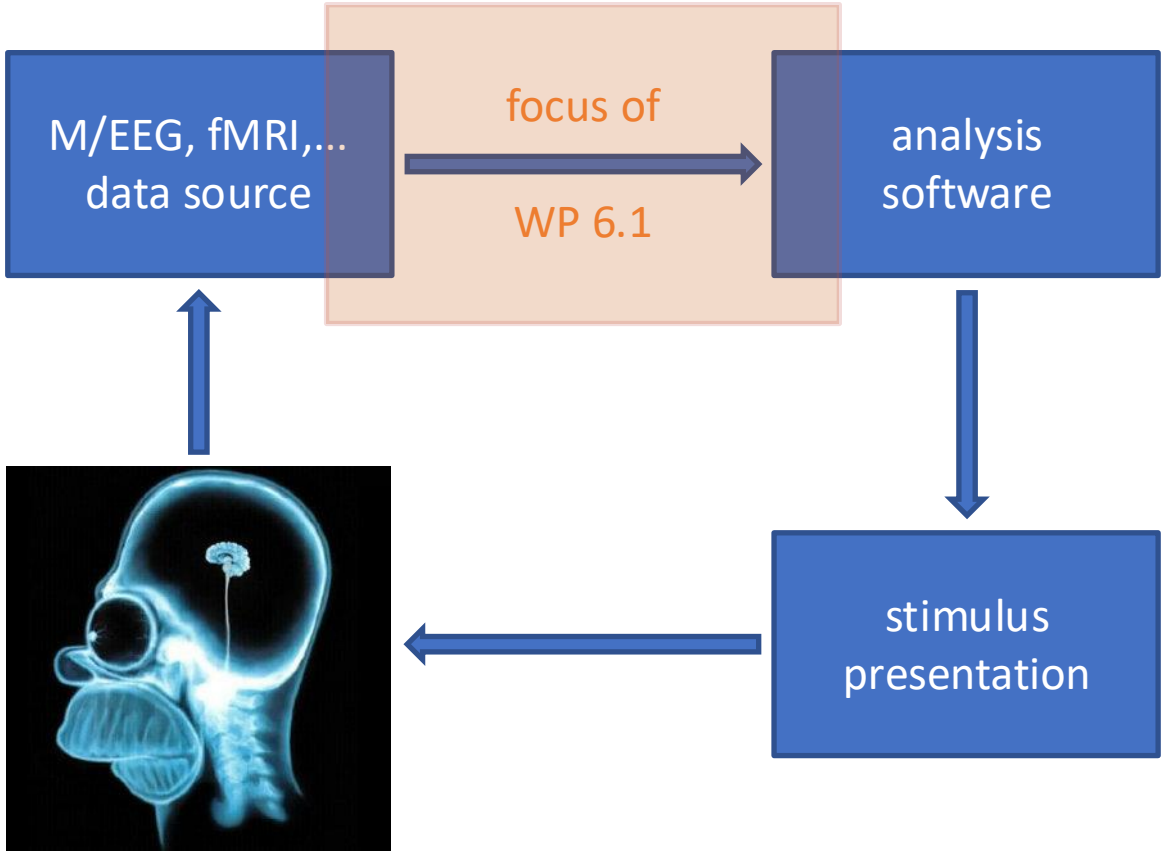
Project 6 structure



Project 6 structure - from real-time data access to an end-user application



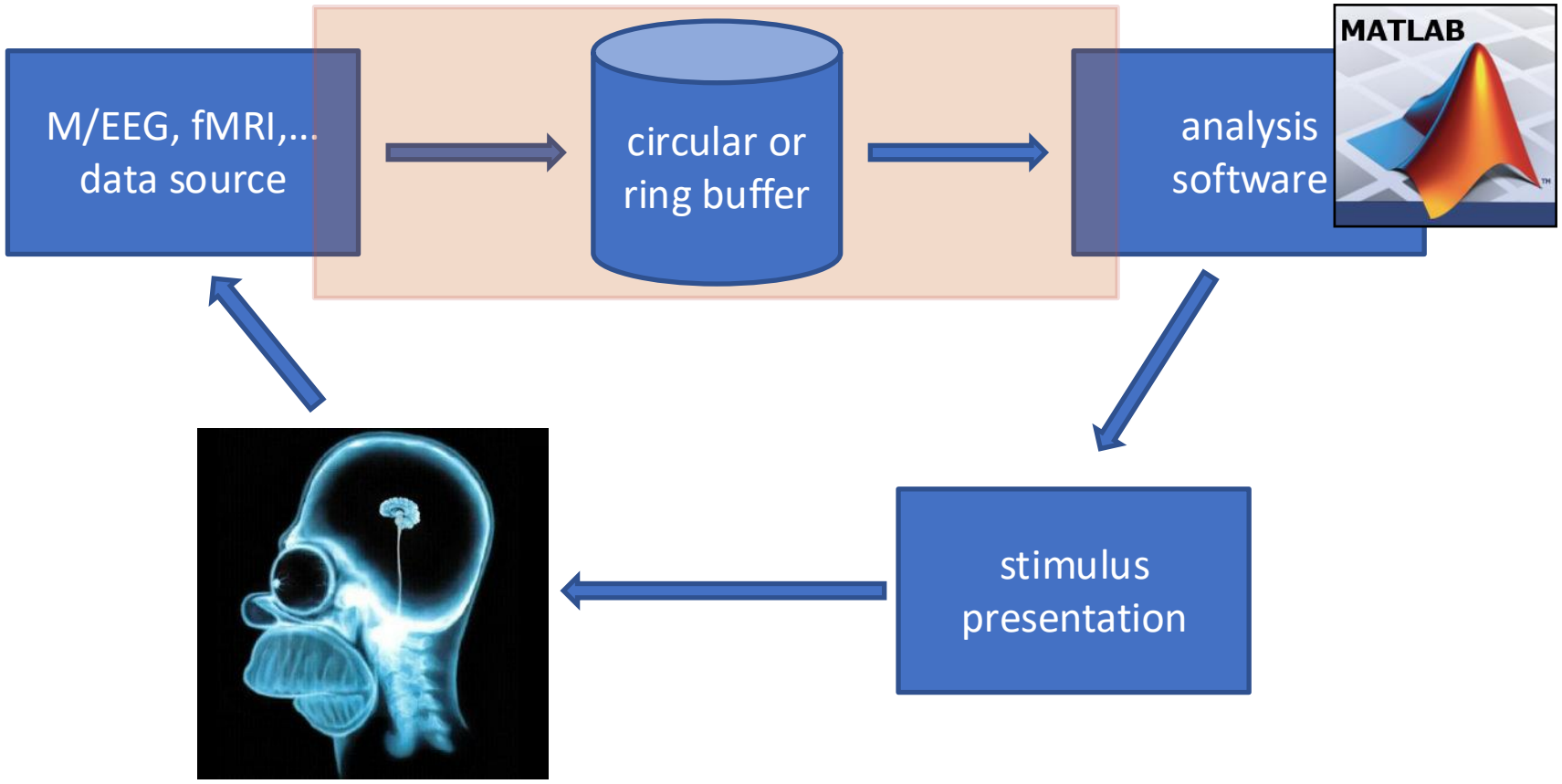
Streaming data in closed-loop systems



need to handle incoming data, processing, sending output

pure streaming protocol is hard to work with

Buffering relaxes timing constraints

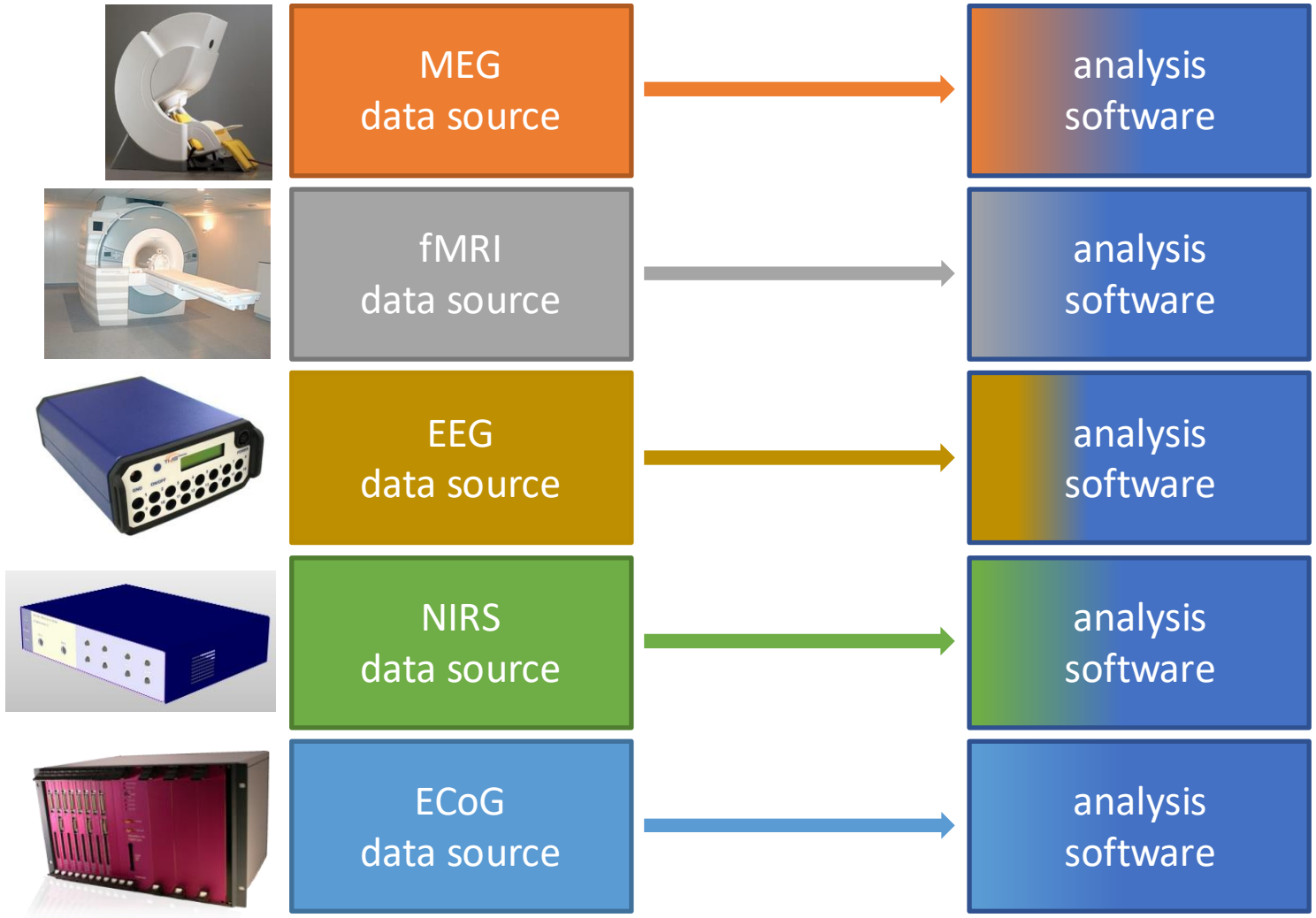


analysis side can fetch data when convenient

can look back in time if needed

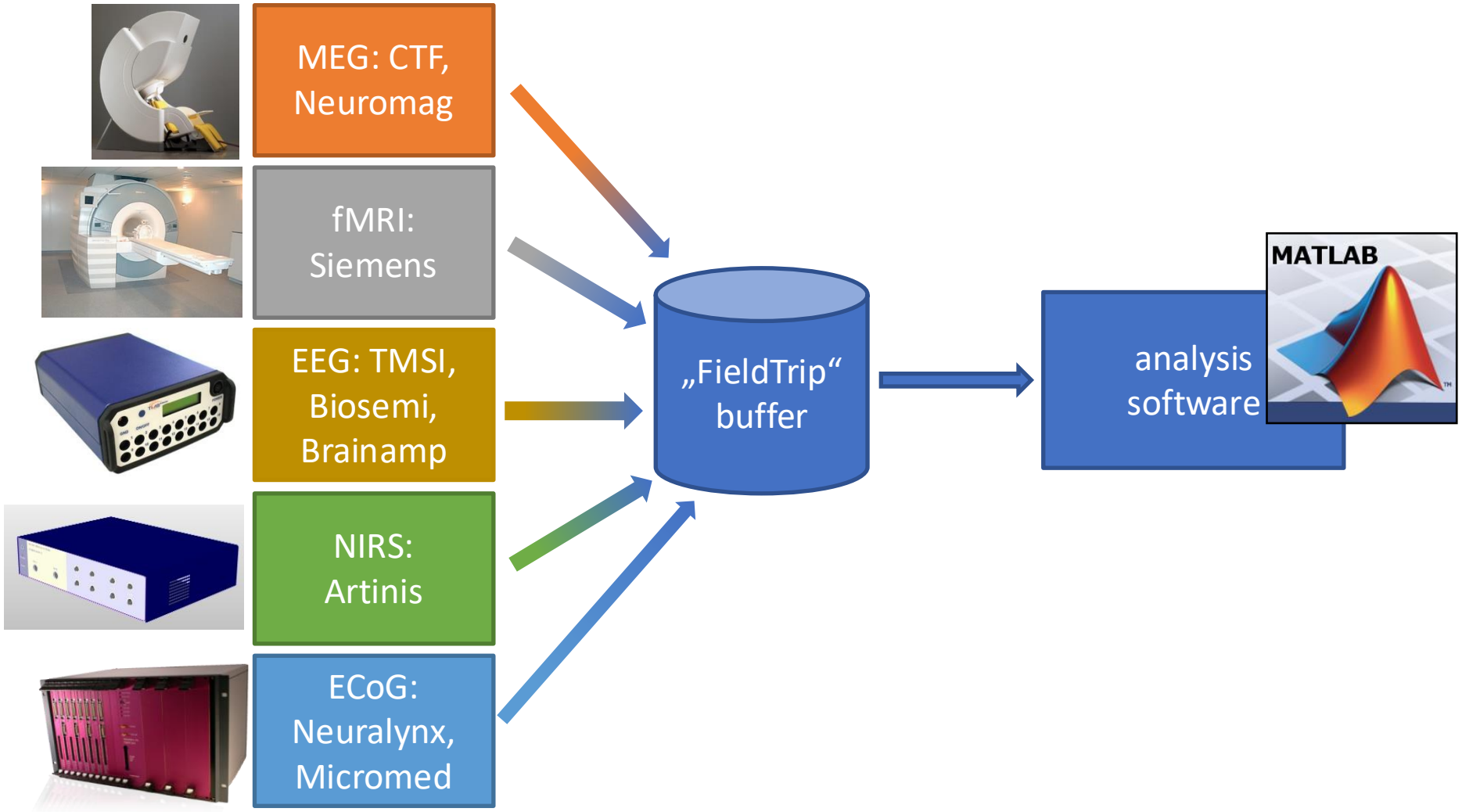
Every device has its own interface

→ lots of device-specific code on analysis side



Use same protocol across devices

→ analysis side gets much simpler



Use same protocol across devices

→ analysis side gets much simpler



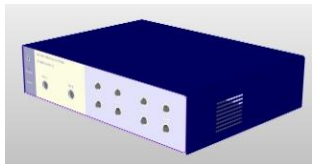
MEG: CTF, Neuromag



fMRI: Siemens



EEG: TMSI, Biosemi, Brainamp



NIRS: Artinis



ECoG: Neuralynx, Micromed

BrainGain provided the system-specific glue

„FieldTrip“ buffer

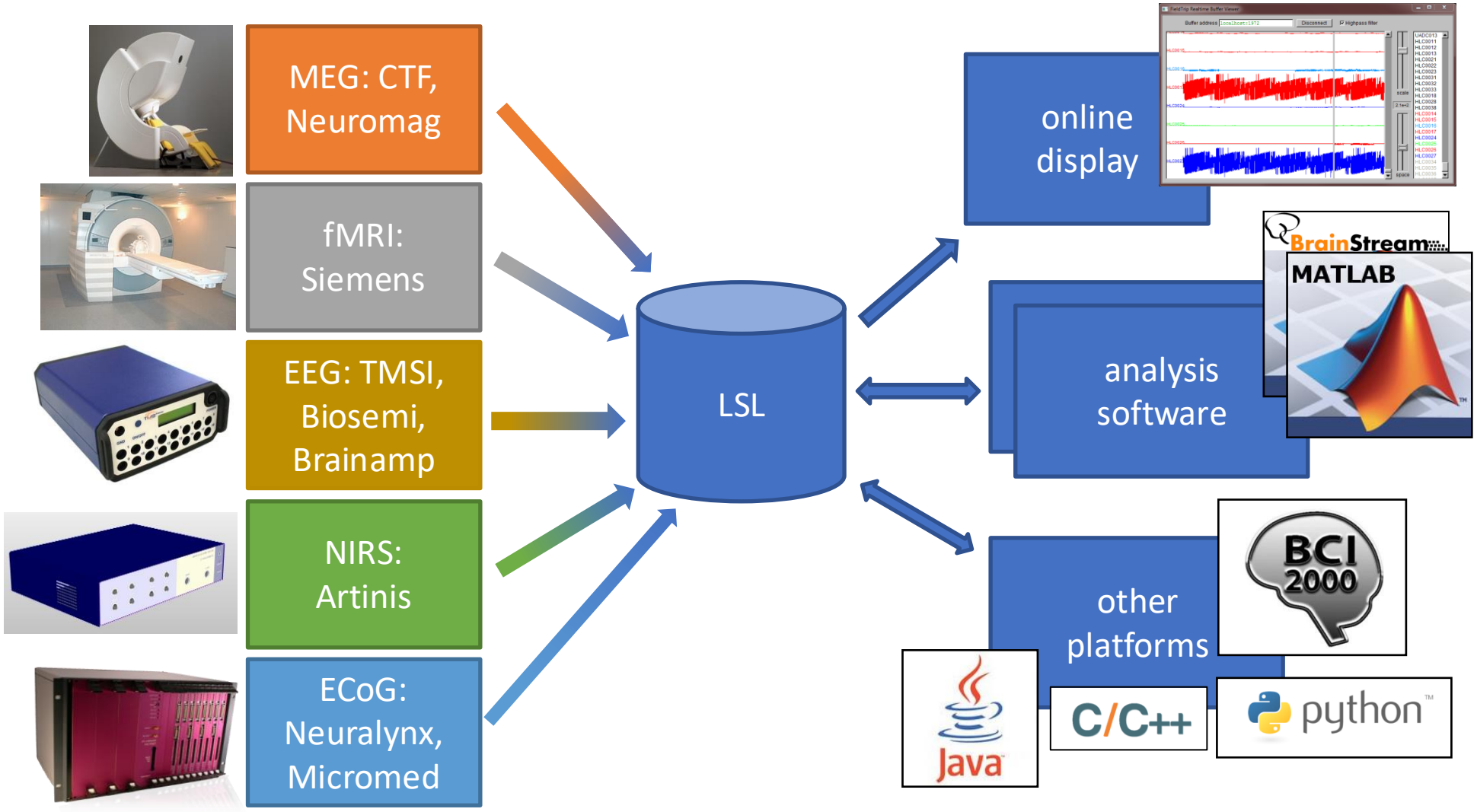
analysis software

MATLAB

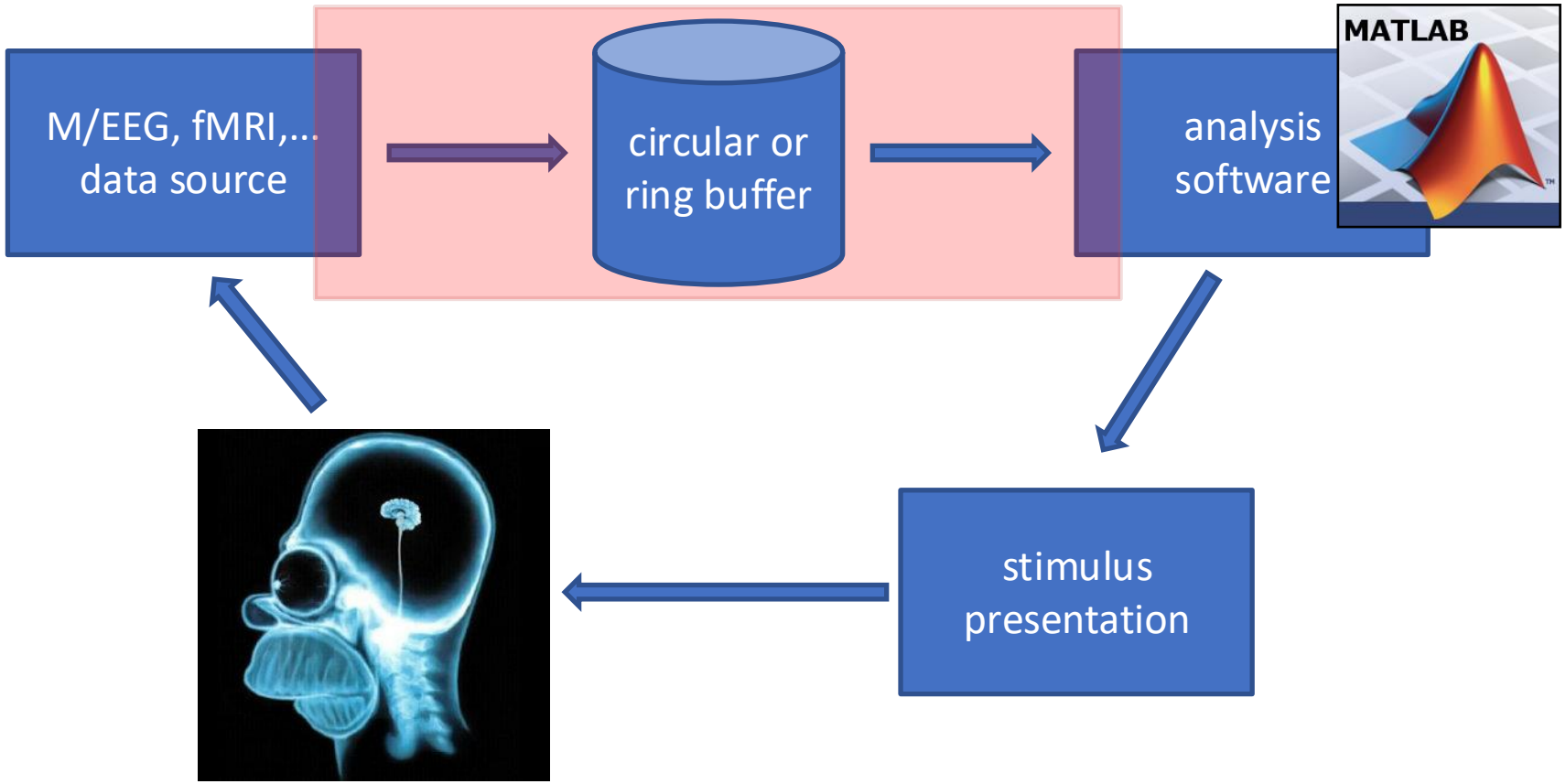
Nowadays everyone uses Lab Streaming Layer (LSL) for this

Multiple applications for analysis etc.

→ stimulus and feedback applications write markers to LSL



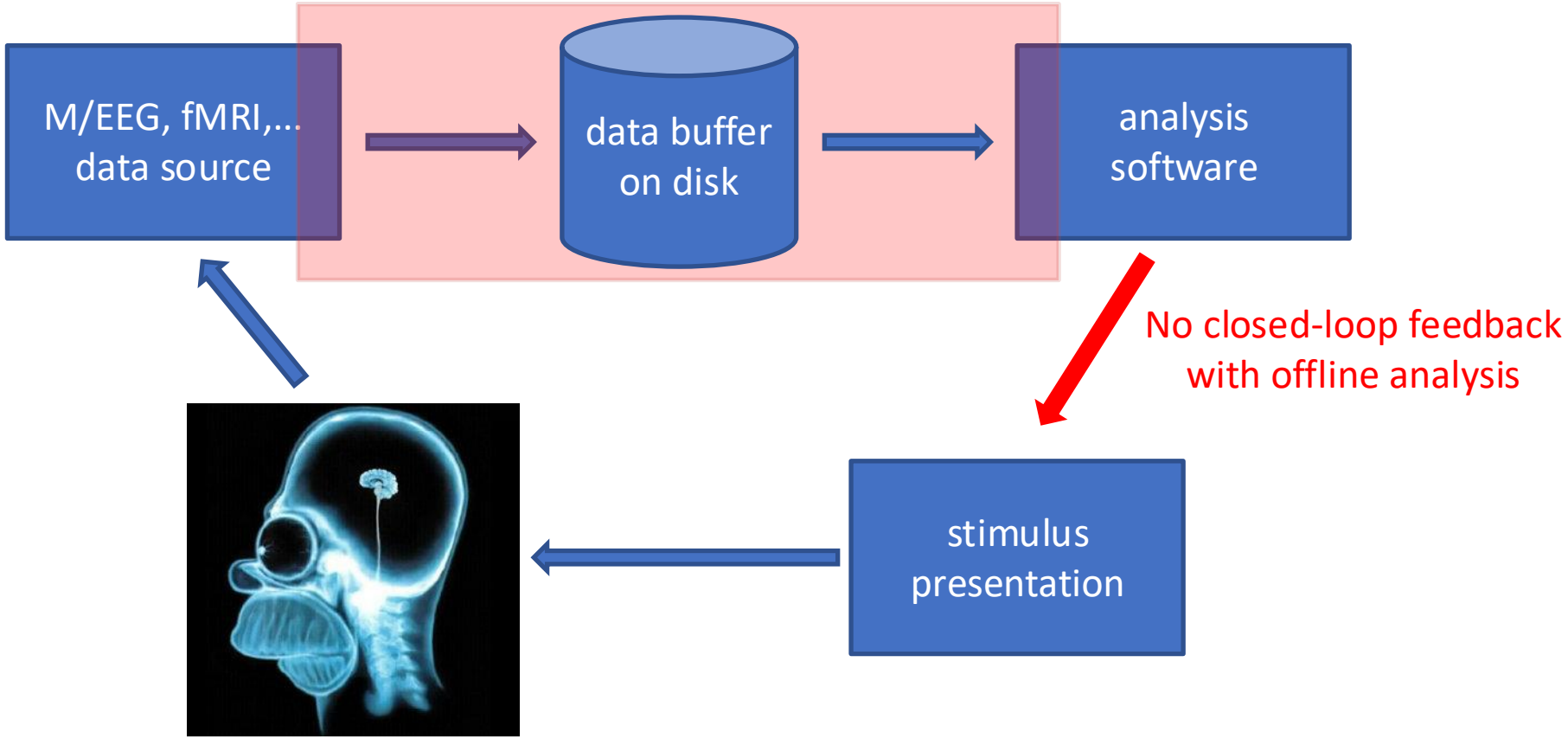
Buffering relaxes timing constraints



analysis software can fetch data when convenient

analysis software can look back in time (if needed)

A recording on disk can also be consisdered a „data buffer“



the data is recorded on day 1

the analysis is performed on day 2

BCI software technology





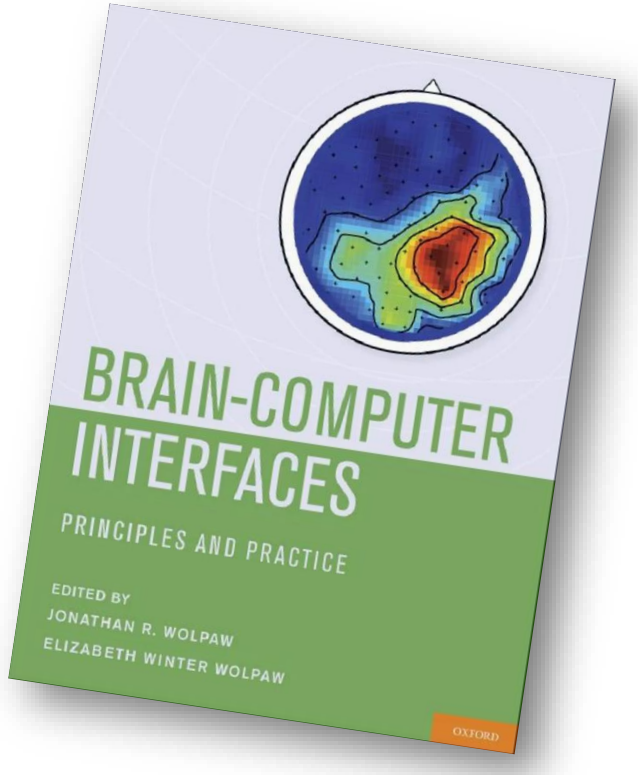


Just for streaming data, platform agnostic

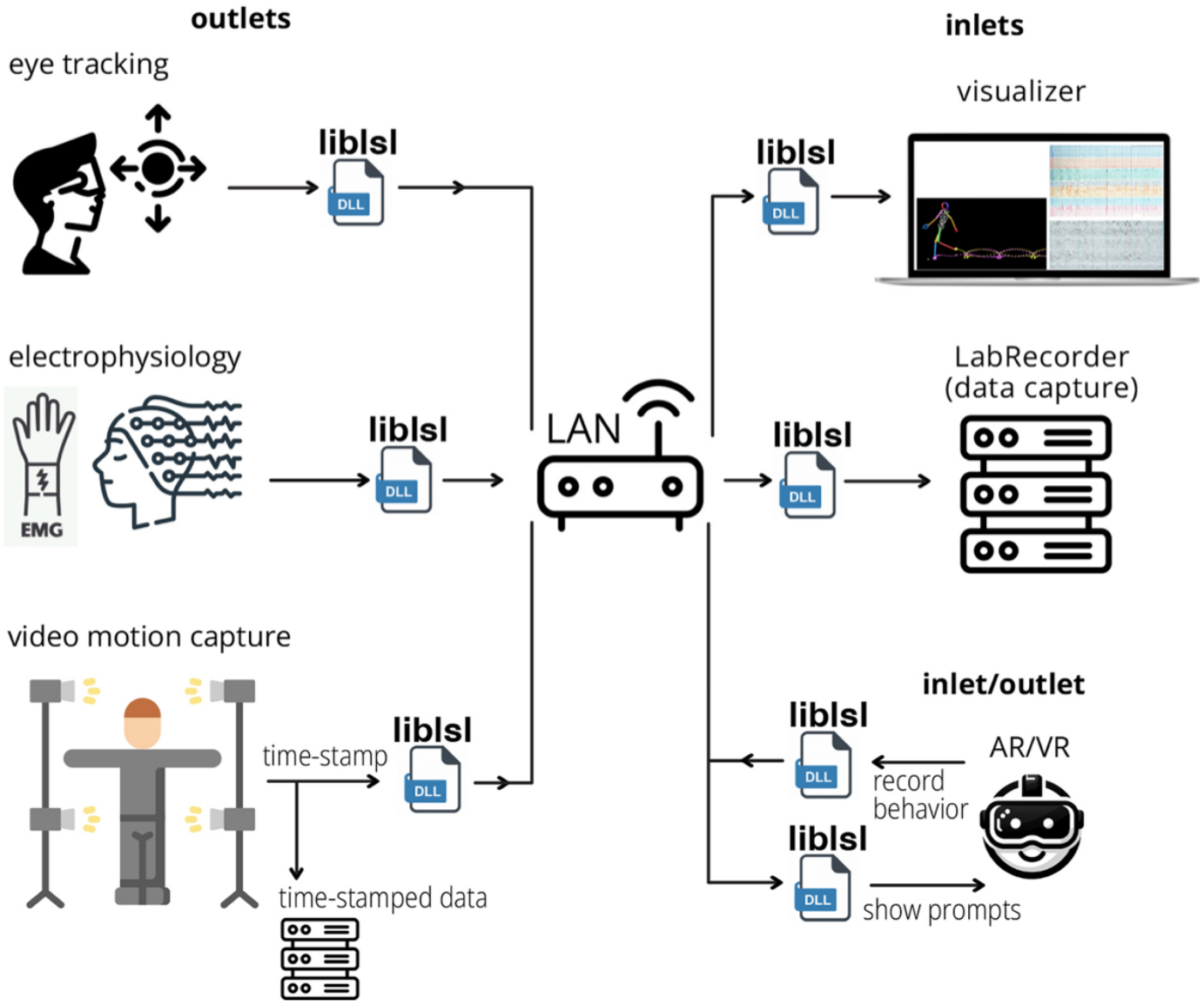
- **FieldTrip** buffer - developed around 2010 for BrainGain
- **TOBI** - EU project around the same time
- Lab Streaming Layer (**LSL**) - USCD project at the same time, nowadays the recommended solution

More complete/full application frameworks

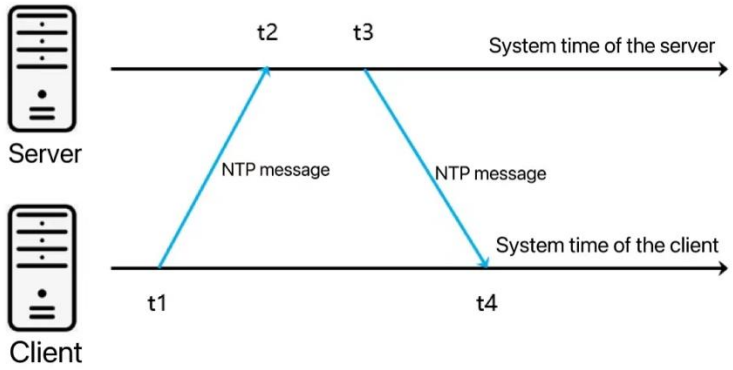
- BCI2000 (Wadsworth Institute, USA)
- OpenVibe (INRIA, France)
- BrainFlow
- BCILAB
- Timeflux
- BciPy, PyBCI
- MNE-CPP



Lab Streaming Layer (LSL)



Lab Streaming Layer (LSL)



Background NTP-like protocol to synchronize the clocks on different computers.

Outlets send ”packets” that include the timestamp of the local clock.

- EEG stream sends small nchan*nsample matrices with a timestamp
- Marker stream sends one timestamped marker at a time
- Video stream saves the video frames locally, but sends the timestamped frame number



Inlets receive the packets (after some network delay) and your real-time software has to deal with the synchronization.

Labrecorder can record various streams and store them together in an XDF file. Upon reading that file, your offline software has to deal with the synchronization.

How “real” is real-time anyway?

Timing requirements

- “real-time” analysis
  - instantaneous
  - very fast
  - well controlled

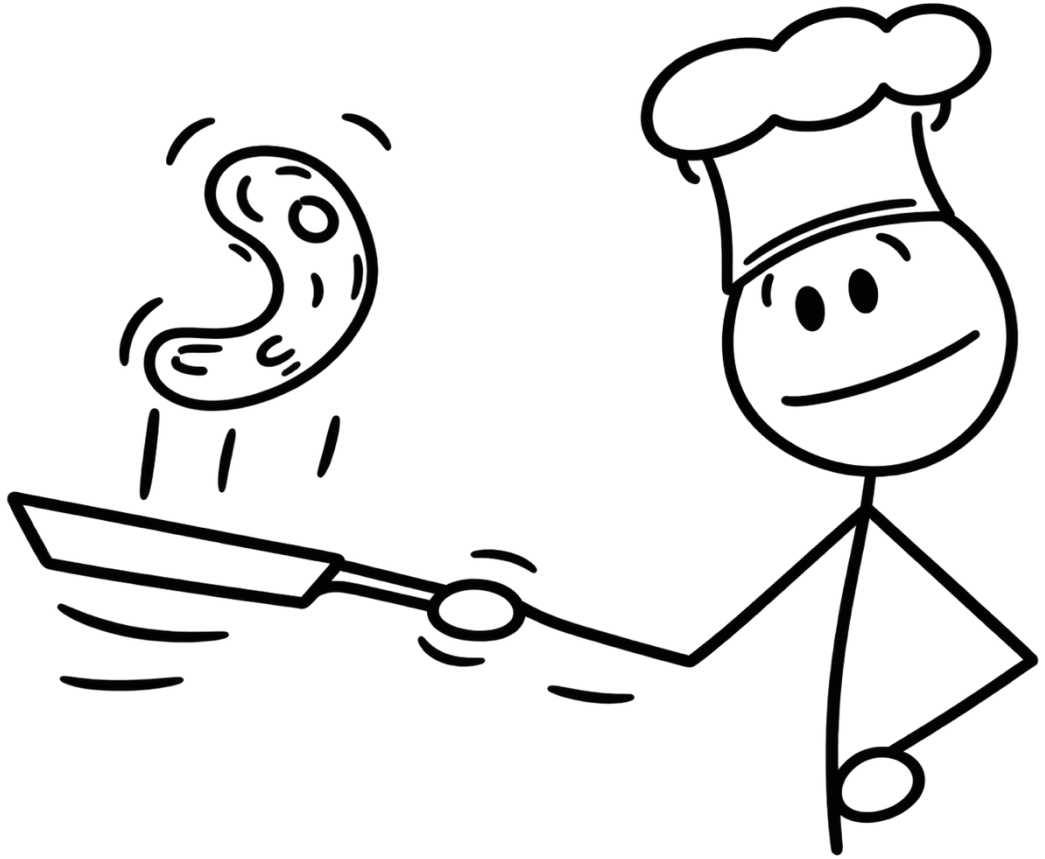
How much time does it take to make pancakes?

Go to the shop to get groceries

Make the pancake batter

Get the pan heated up

And then: 3 minutes for each pancake



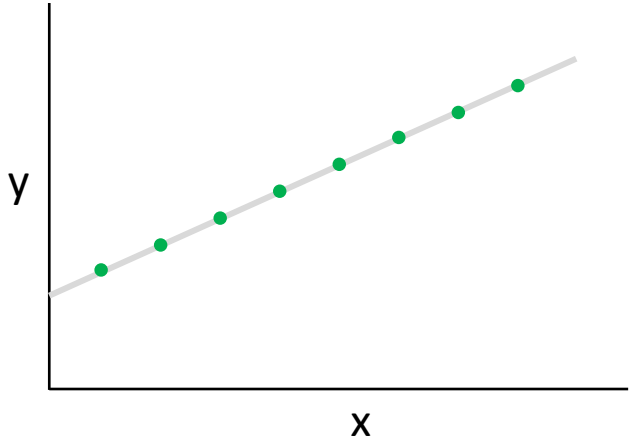
Timing requirements

"real-time" analysis

- instantaneous
- very fast
- well controlled

aspects that influence timing

- offset
- delay
- jitter



y = a*x + b

time = offset + delay

time = offset + complexity*datasize

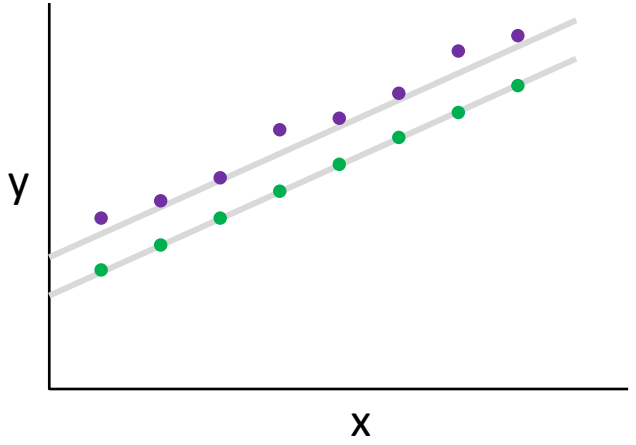
Timing requirements

"real-time" analysis

- instantaneous
- very fast
- well controlled

aspects that influence timing

- offset
- delay
- jitter



y = a*x + b

time = offset + delay + jitter

time = offset + complexity*datasize + jitter

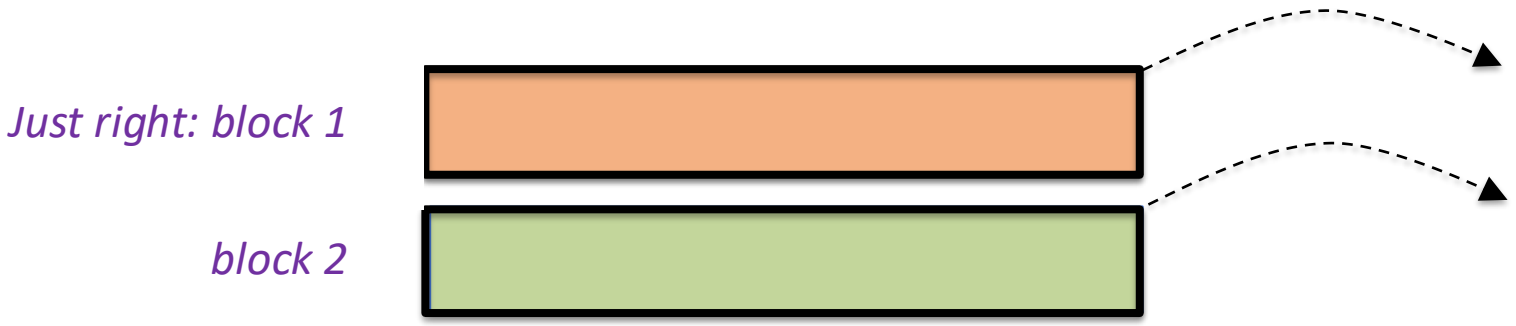
Timing requirements

- Minimize the offset
  - hardware
  - architectural choices
- Minimize the delay
  - reduce data size
  - increase speed of algorithm
  - increase bandwith of transfer
- Minimize the jitter

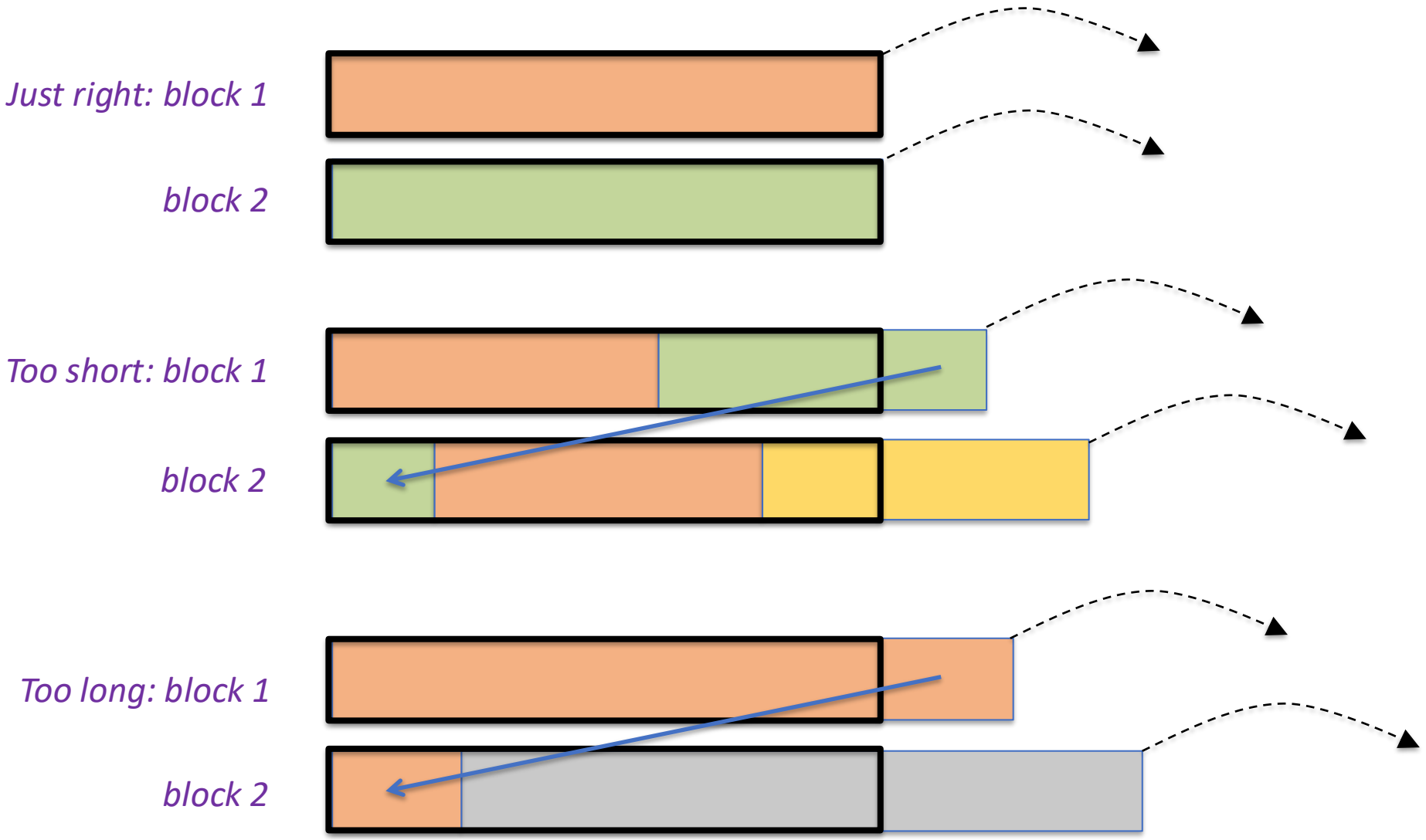
Understanding jitter and blocksize



Understanding jitter and blocksize



Understanding jitter and blocksize



Understanding jitter and blocksize

data is processed in blocks everywhere to increase *efficiency* and *bandwith*

- inside the EEG amplifier
- serial interface over USB or Bluetooth
- TCP packet over the wired or wifi network
- processing stream step 1
- processing stream step 2

unequal block sizes cause jitter



Timing requirements

Why not always use blocks of 1 sample?

for 1 sample

delay = speed*datasize + offset

for N times a 1-sample block

delay = N*speed*datasize + N*offset

for N samples in one block

delay = N*speed*datasize + offset

Algorithms for efficient analyses

What is an efficient computation?

- computes in well-defined time
- time does not increase

As an example

- compute the running mean, i.e. the mean of all data up to now

Running mean

```
n  = 0;


while (true)

        hdr = ft_read_header(buffer);

        if hdr.nSamples<=n
                continue;
        end

        dat = ft_read_data(buffer, 'begsample', 1, 'endsample', hdr.nSamples);
        avg = mean(dat,2);


        n = hdr.nSamples;



end
```

Running mean

```
n  = 0;
s  = zeros(nchan,1);


while (true)

        hdr = ft_read_header(buffer);

        if hdr.nSamples<=n
                continue;
        end

        dat = ft_read_data(buffer, 'begsample', n+1, 'endsample', hdr.nSamples);
        s   = s  +  sum(dat,2);


        n   = hdr.nSamples;
        avg = s/n;


end
```

Algorithms for efficient analyses

- what is efficient?
  - computes in well-defined time
  - use only short snippets of streaming data
  - incremental algorithms
- what might be problematic?
  - algorithms that take variable or unknown time
  - iterative convergence

Algorithms for efficient analyses

- combine linear operations
- example
  - spectral power of spatial filter output

Spectral power of spatial filter output

```
w = randn(n,n); % say that w is a N*N common-spatial-patterns (CSP) unmixing matrix


while (true)

          hdr = ft_read_header(buffer);
          s1 = hdr.nSamples - 100;
          s2 = hdr.nSamples;

          dat = ft_read_data(buffer, 'begsample, s1, 'endsample', s2);

          ampl = fft(w * dat);  % compute the complex-valued amplitude spectrum

          ampl = ampl(compindx, :);
          ampl = ampl(:, freqindx);

          pow = abs(ampl).^2;

end
```

Spectral power of spatial filter output

```
w = randn(n,n); % say that w is a N*N common-spatial-patterns (CSP) unmixing matrix


while (true)

          hdr = ft_read_header(buffer);
          s1 = hdr.nSamples - 100;
          s2 = hdr.nSamples;

          dat = ft_read_data(buffer, 'begsample, s1, 'endsample', s2);

          ampl = w * fft(dat);

          ampl = ampl(compindx, :);
          ampl = ampl(:, freqindx);

          pow = abs(ampl).^2;

end
```

Spectral power of spatial filter output

```
w = randn(n,n); % say that w is a N*N common-spatial-patterns (CSP) unmixing matrix


while (true)

          hdr = ft_read_header(buffer);
          s1 = hdr.nSamples - 100;
          s2 = hdr.nSamples;

          dat = ft_read_data(buffer, 'begsample, s1, 'endsample', s2);

          ampl = w(compindx,:) * fft(dat);


          ampl = ampl(:, freqindx);

          pow = abs(ampl).^2;

end
```

Spectral power of spatial filter output

```
w = randn(n,n); % say that w is a N*N common-spatial-patterns (CSP) unmixing matrix

f = [cos(2*pi*t/foi) + i*sin(2*pi*t/foi)]';

while (true)

          hdr = ft_read_header(buffer);
          s1 = hdr.nSamples - 100;
          s2 = hdr.nSamples;

          dat = ft_read_data(buffer, 'begsample, s1, 'endsample', s2);

          ampl = w(compindx,:) * dat * f;



          pow = abs(ampl).^2;

end
```

fft = O[n log(n)]
dft = O[n]

Spectral power of spatial filter output

```
w = randn(n,n); % say that w is a N*N common-spatial-patterns (CSP) unmixing matrix
w = w(compindx,:);
f = [cos(2*pi*t/foi) + i*sin(2*pi*t/foi)]';

while (true)

          hdr = ft_read_header(buffer);
          s1 = hdr.nSamples - 100;
          s2 = hdr.nSamples;

          dat = ft_read_data(buffer, 'begsample, s1, 'endsample', s2);

          ampl = w * dat * f';



          pow = abs(ampl).^2;

end
```

Algorithms for efficient analyses

Spectral estimation
- do not convolve!
- discrete (DFT) instead of “fast” fourier transform (FFT)
- use Welch method instead of multitapering
- use autoregressive (AR) model estimate

Regression or general linear estimation (GLM)
- requires inverse of covariance matrix
- use incremental “small rank adjustment”

Reduce data size as early as possible

Move operations outside the for- or while-loop

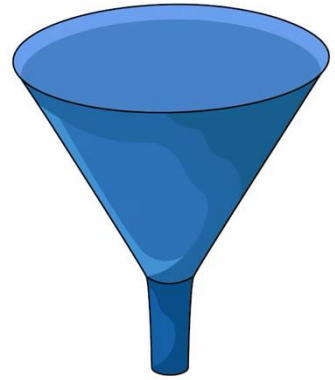
Avoid polling, use interrupts

…

Pipeline sequence



raw data



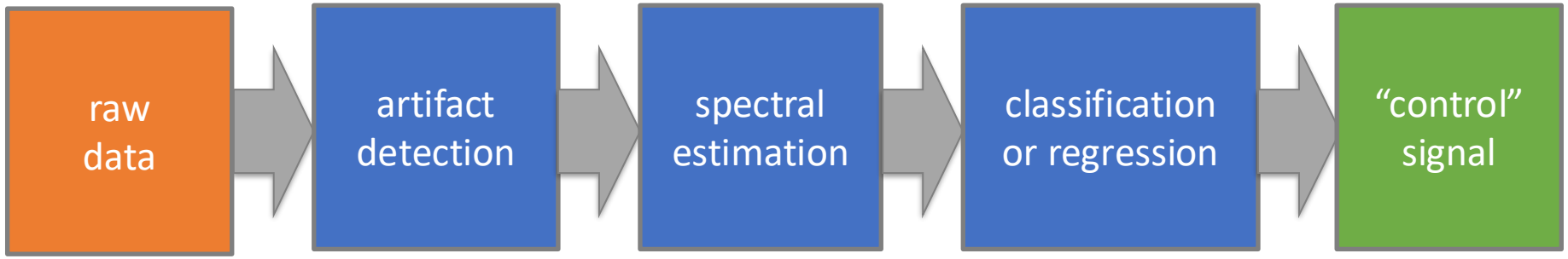


control signal

Building blocks for analyses

- matrices, vectors, linear algebra
- non-linear operations

Combine them in a pipeline that transform raw data into a control signal



Closing the loop

Asynchronous BCI (user-paced)

- Raw data comes in
- Control signal goes out
- For example seizure detection and alarm

Synchronous BCI (system-paced)

- Stimuli are presented to the subject
- Brain response to the stimuli is detected
- Stimuli have to be updated accordingly
- For example P300 speller



Closing the loop

Sharing data/markers within computers

- single multithreaded application
- shared memory
- sockets

Sharing data/markers between computers

- serial port (USB)
- parallel port or digital I/O card
- ethernet (TCP, UDP)
- key-value store (like Redis)
- mqtt, ZeroMQ

Platform considerations

Low-level programming environment

Compiled languages like C/C++ are fast at run-time
but slow during development.

High-level with Python, MATLAB, Julia, etc.

Interpreted languages are fast during development
and can be fast enough during execution

Integrated BCI application (like BCI2000, OpenVibe, or EEGsynth)
allow components to be reused but have a longer learning curve.

General considerations

- Time is *the* limiting factor
  - CPU time

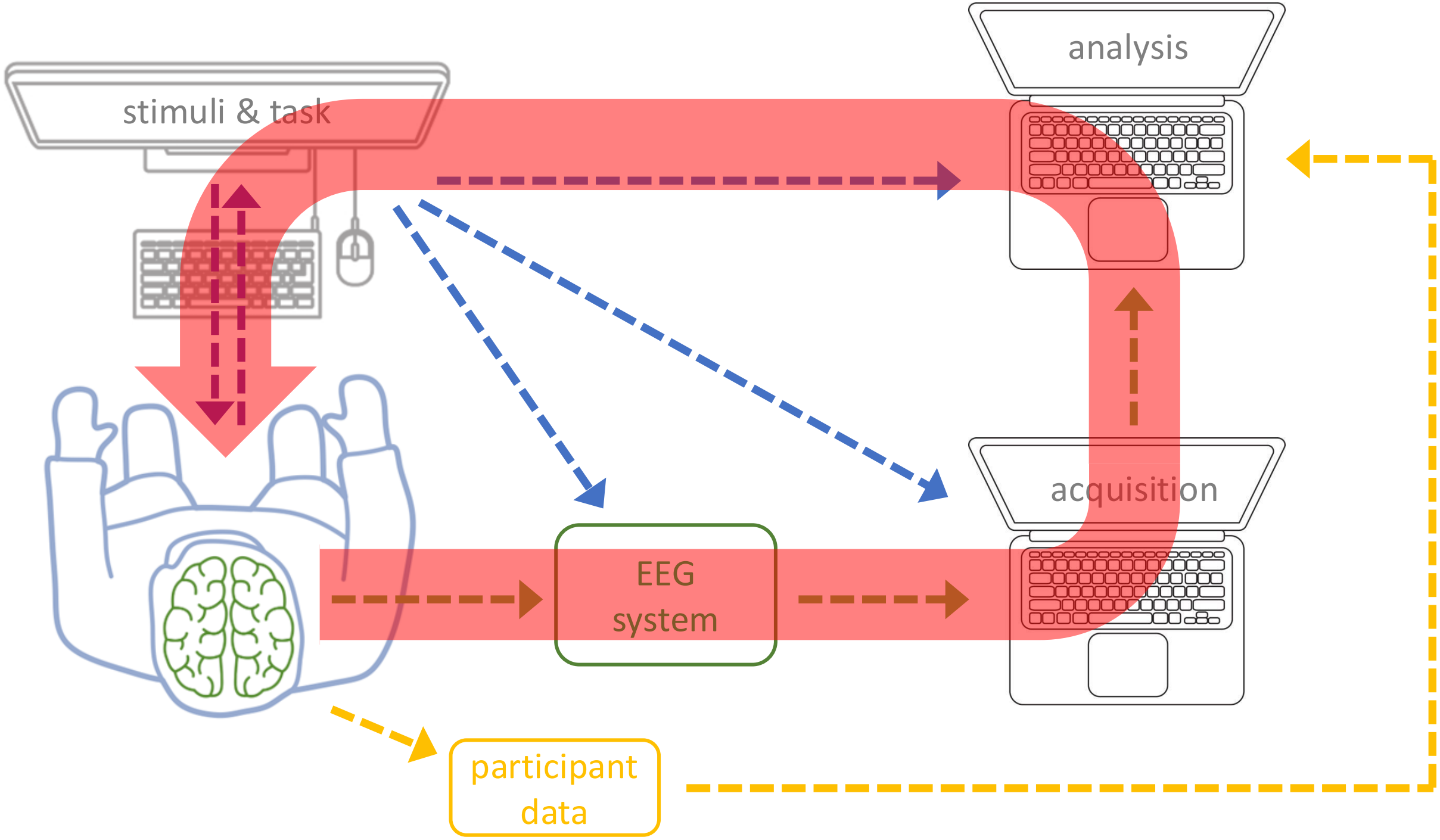
General considerations

Time is *the* limiting factor

~~CPU~~ your time

There is not only a closed-loop cycle for the BCI system; there is also the closed-loop in rapid-application development.

Make a prototype, test, update the prototype, test, …


analysis
stimuli & task
acquisition
EEG
system
participant
data