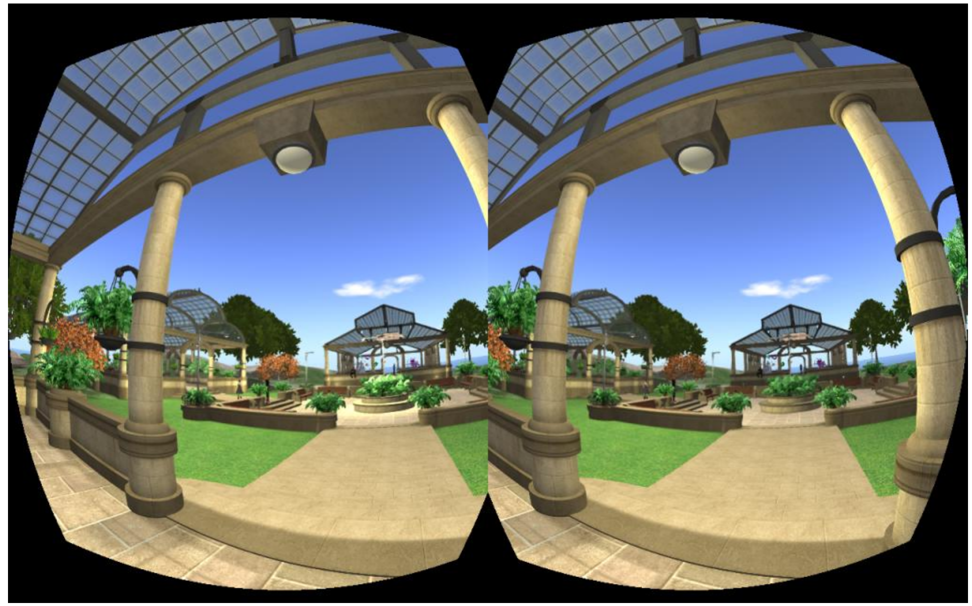
Performance: Metrics and Scales

- **Metric**: a way of measuring or characterising a system
  – May use “metric”, “measure”, or “property” depending on context
- **Scale**: the unit of measurement for a given metric
  – If you apply the metric, what sort of result does it give?
- Need consistent metrics within the design process
  – Requirements: *what properties should the solution have?*
  – Analysis: *what space of solutions meet the requirements?*
  – Design: *how best to implement the identified solution?*
  – Evaluation: *does the solution meet requirements?*
  – Optimisation: *can the solution be made any better?*
- Need metrics to compare multiple solutions
  – *Is system A better than system B?*

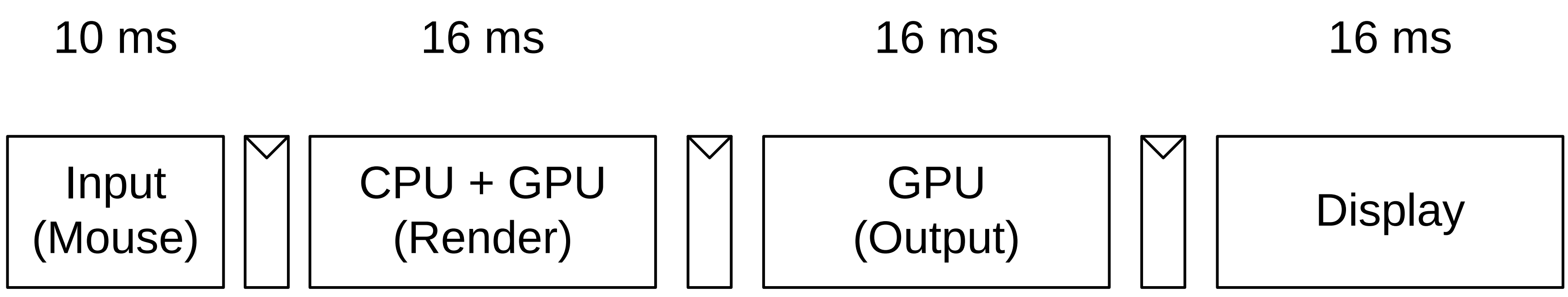
Example: Oculus Rift



All images owned by Oculus VR, as best I can tell

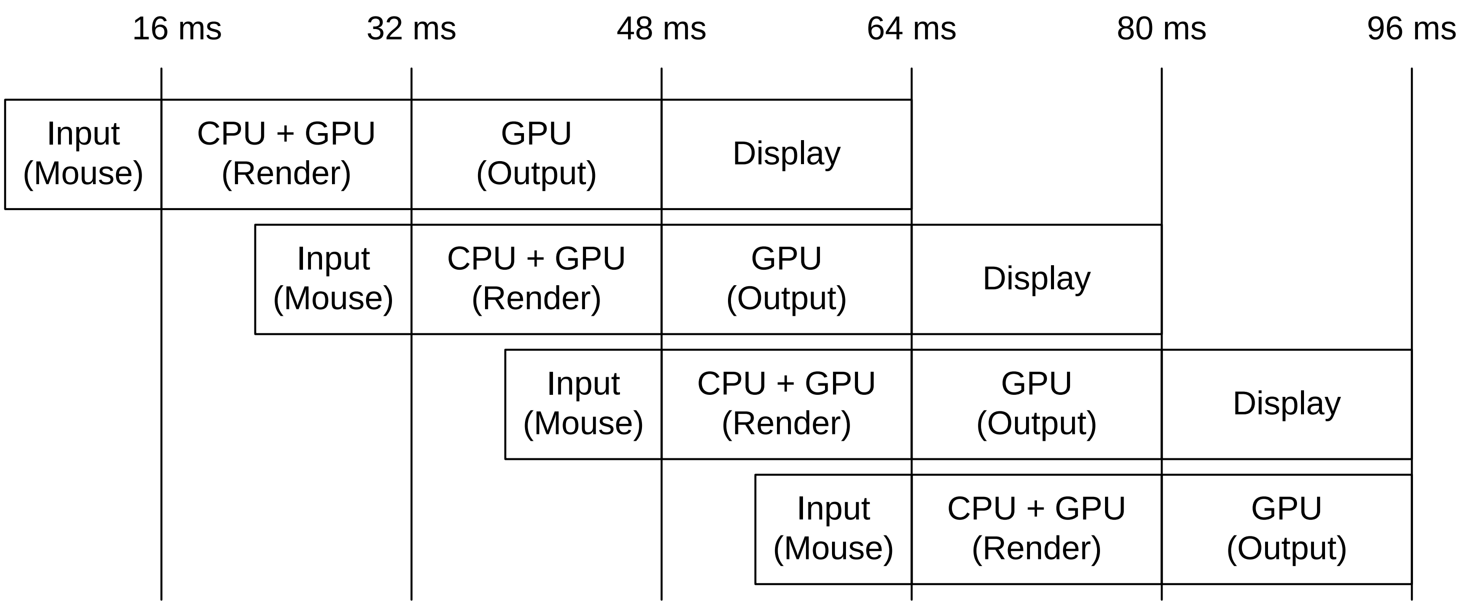
Typical graphics pipeline

- Most games can get 60Hz to 120Hz
  - 60Hz is usually enough for perceived smooth motion
- Results are buffered between each stage
  - Each stage has a throughput of 60Hz+
  - Takes some time to scan data from GPU RAM to display
  - Displays have persistence; brightness change not instantaneous



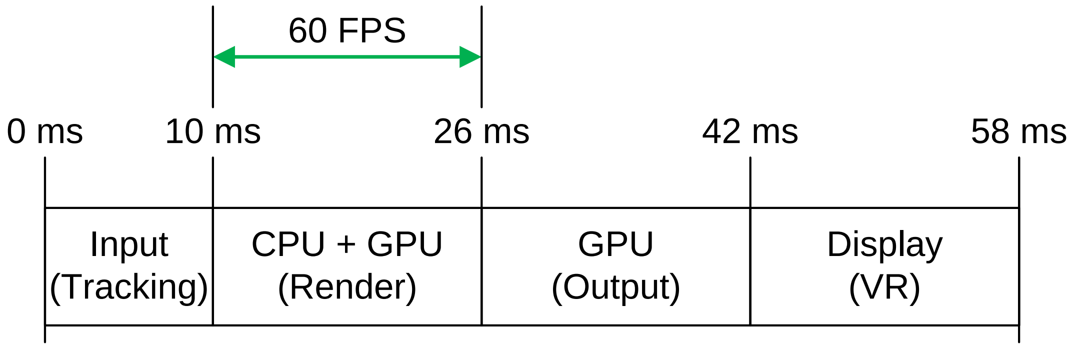
Typical graphics pipeline

- Classic pipeline: *performance is limited by the slowest stage*
- Once the pipeline is primed, we can get one frame per cycle



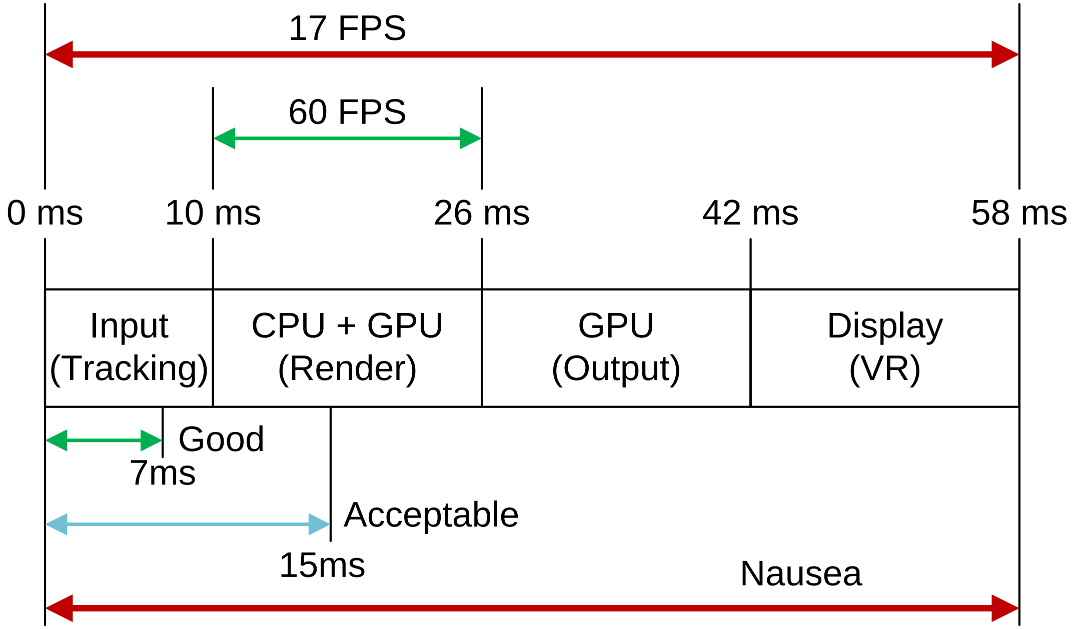
VR puts a constraint on latency

- Feedback loop: body->tracking->GPU->display->eye



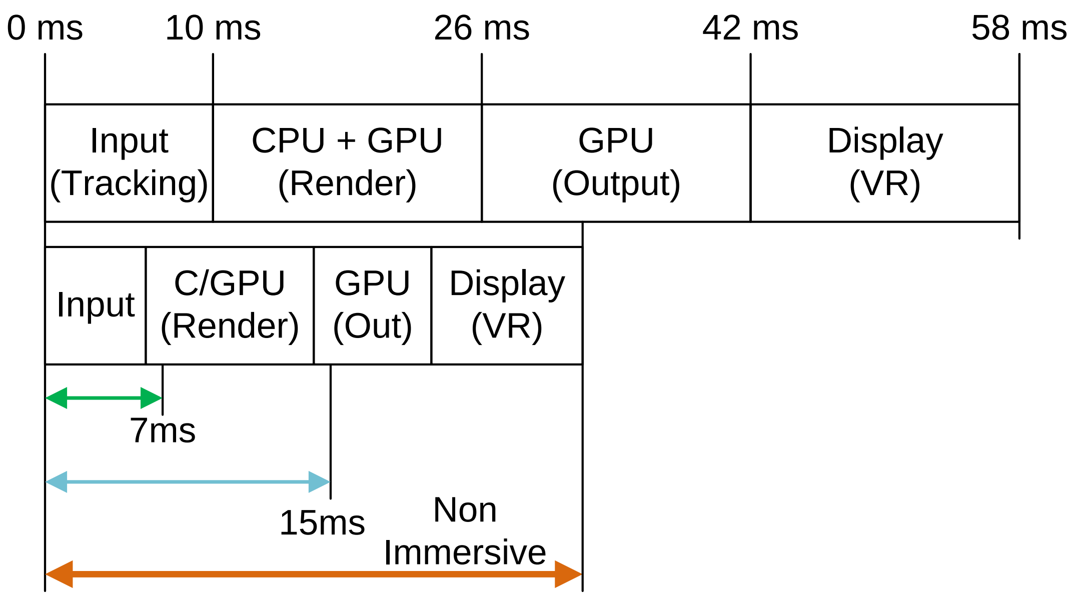
VR puts a constraint on latency

- Feedback loop: body->tracking->GPU->display->eye
- Humans are very sensitive to latency discrepancies
  - Mismatch between movement and vision: “*I’m poisoned*”!
  - Need 7-15ms for good user experience



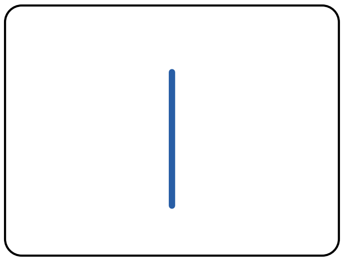
Some solutions in hardware and software

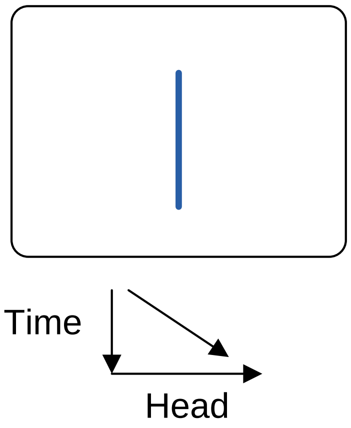
- A number of things can reduce overall complexity
  - Reduce scene complexity and pixel count (late 90s quality)
  - Better input hardware with low latency
  - Different output technology: e.g. laser scanning displays
- Still difficult to get below 25ms (*even though 40FPS is trivial*!)



Remaining problem: distortion

- Images are displayed in raster scan order
- Both eyes and the head are moving independently

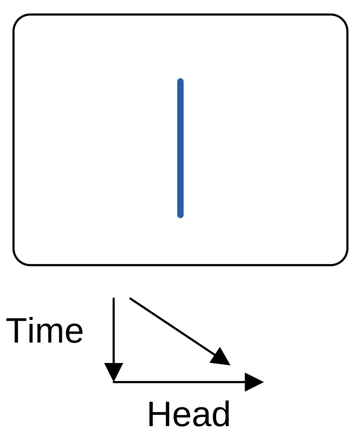


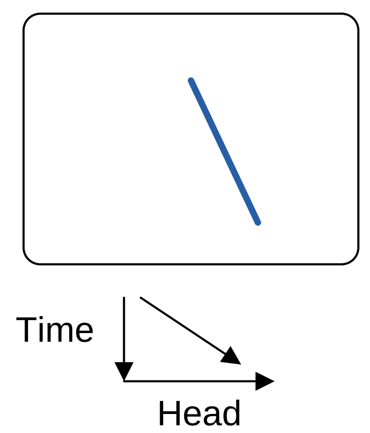


Michael Abrash:
http://blogs.valvesoftware.com/abrash/raster-scan-displays-more-than-meets-the-eye/

Remaining problem: distortion

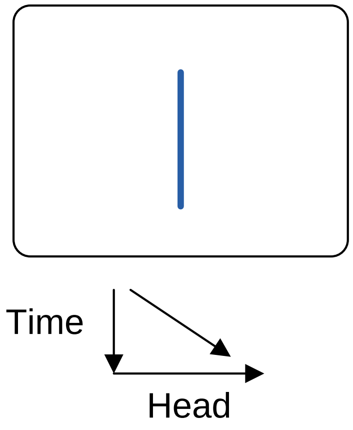
- Images are displayed in raster scan order
- Both eyes and the head are moving independently
  - Perceived object is different from displayed object

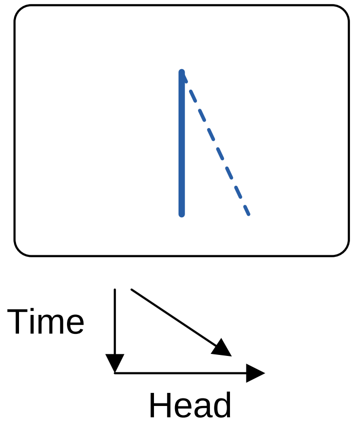




Remaining problem: distortion

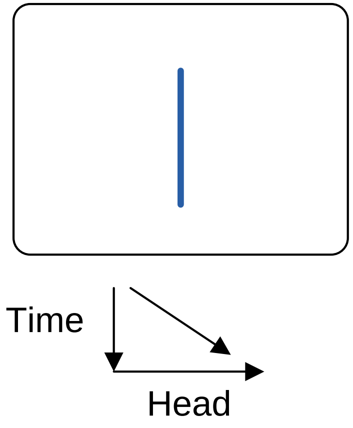
- Images are displayed in raster scan order
- Both eyes and the head are moving independently
  - Perceived object is different from displayed object
- Can try to pre-warp the geometry according to motion

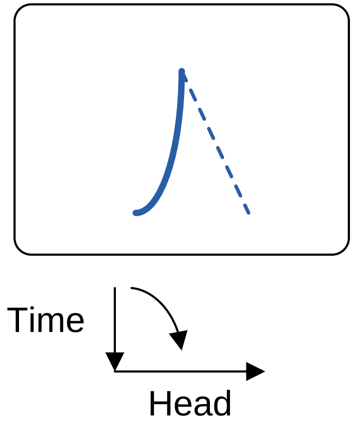




Remaining problem: distortion

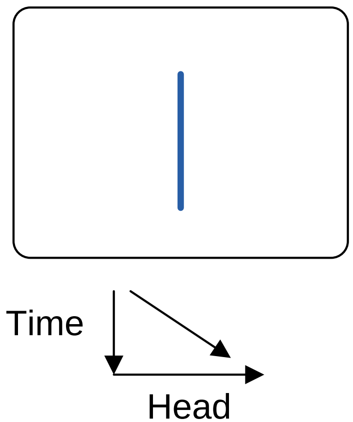
- Images are displayed in raster scan order
- Both eyes and the head are moving independently
  - Perceived object is different from displayed object
- Can try to pre-warp the geometry according to motion
  - But we only update at 17 FPS: motion can change

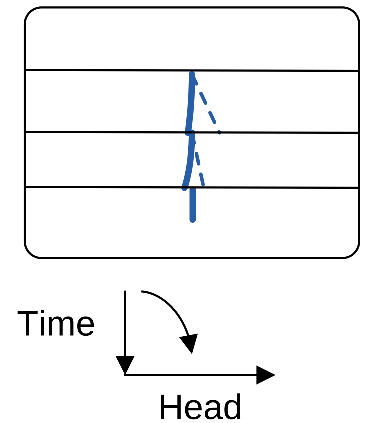




Partial solution: *race the beam*

- Render image in horizontal chunks
  - Each chunk uses the very latest tracking information
  - Chunks are sent to display as soon as they are ready
- The smaller the chunks, the less perceived distortion





“Performance” is application dependent

- Throughput and latency are related, but very different
  - Throughput of X -> latency >= 1/X
  - Latency of L -> throughput = 1/L (usually)
- Usually quite easy to get high throughput
  - Throughput of 10 GFlops vs latency of 100 ps
  - Throughput of 1000 FPS vs latency of 1ms per image
- People try to optimise one to improve the other
  - Increase mouse update speed
  - Tweak GPU settings to get 120Hz
  - (Do they have any effect?)

Qualitative vs Quantitative metrics

- **Quantitative**: hard numbers, measurable
  - Time, Energy, Space
  - Signal-to-Noise, Frames-per-second, Memory Usage
  - Money (?)
- **Qualitative**: feelings, opinions
  - Complexity: “Simple”, “Tricky”
  - Design Effort: “Easy”, “Hard”
  - User Experience: “snappy”, “intuitive”, “pretty”

- Try to use quantitative measurements when possible
  - Repeatable: all measurements yield the same result
  - Verifiable: third-parties should get the same result

Types of scale

- **Nominal**: categories with no intrinsic order
  - Apples, Oranges, Pears
- **Ordinal**: categories which can be compared
  - Bad, Ok, Good
- **Interval**: numbers with meaningful differences
  - Year
- **Ratio**: numbers with differences and a meaningful zero
  - Frames per second

Choosing scales

- A metric can be defined in terms of many scales
- Temperature
  - *Nominal*: “Balmy”, “Temperate”
  - *Ordinal*: Cold, Chilly, Warm, Hot
  - *Interval*: Degrees Fahrenheit
  - *Ratio*: Degrees Kelvin
- Can create mapping from ratio to other scales
  - Fahrenheit = 9 * K / 5 − 459.67
  - Cold = {K<280}, Chilly = {K>=280 && K<=288}, ...
  - Balmy = {K>293 && K<300}; Temperate = {K>288 && K<=293}
- Can’t go from nominal to ratio without inventing information
  - “Cold” represents a range, but Kelvin is a point

The joys of ratio scales

- Easy to see which system is better
  - If metric(A) < metric(B) then B is better than A
- Easy to see *by how much* the system is better
  - A is better than B by a factor metric(A)/metric(B)
- We can define meaningful combination metrics
  - metric(X) = max[metric(A),metric(B)]
  - metric(Y) = metric(A) + metric(B)
  - metric(Z) = sqrt[metric(A)^2 + metric(B)^2]

Averaging of ratio metrics

- Many metrics are formed from a number of measurements
  - Benchmarks: perform the same task on many different inputs
  - Smoothing: remove statistical noise from different runs
- Ratio metrics can be meaningfully averaged
  - But usually ***not*** using the arithmetic mean
- Geometric mean g(x)
  - Good for combining values on different scales
  - Useful for summarising improvements
- Harmonic mean h(x)
  - Sometimes useful for averaging rates
  - *Usually better to stick to geometric*

$$a(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$g(\mathbf{x}) = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

$$h(\mathbf{x}) = \frac{n}{\sum_{i=1}^{n} 1/x_i}$$

$$h(\mathbf{x}) \le g(\mathbf{x}) \le a(\mathbf{x})$$

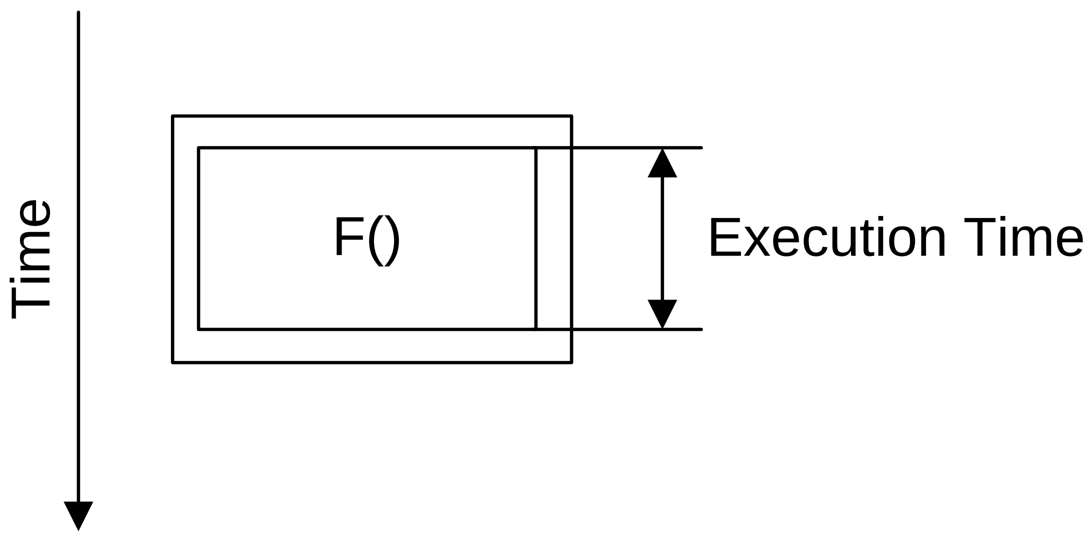
"How not to lie with statistics: the correct way to summarize benchmark results", P. Fleming, J. Wallace, 1986, http://dx.doi.org/10.1145%2F5666.5673

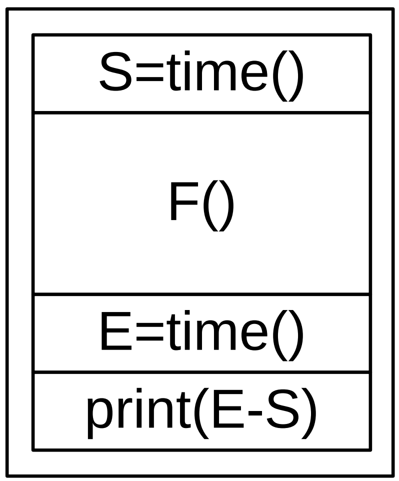
Execution Time

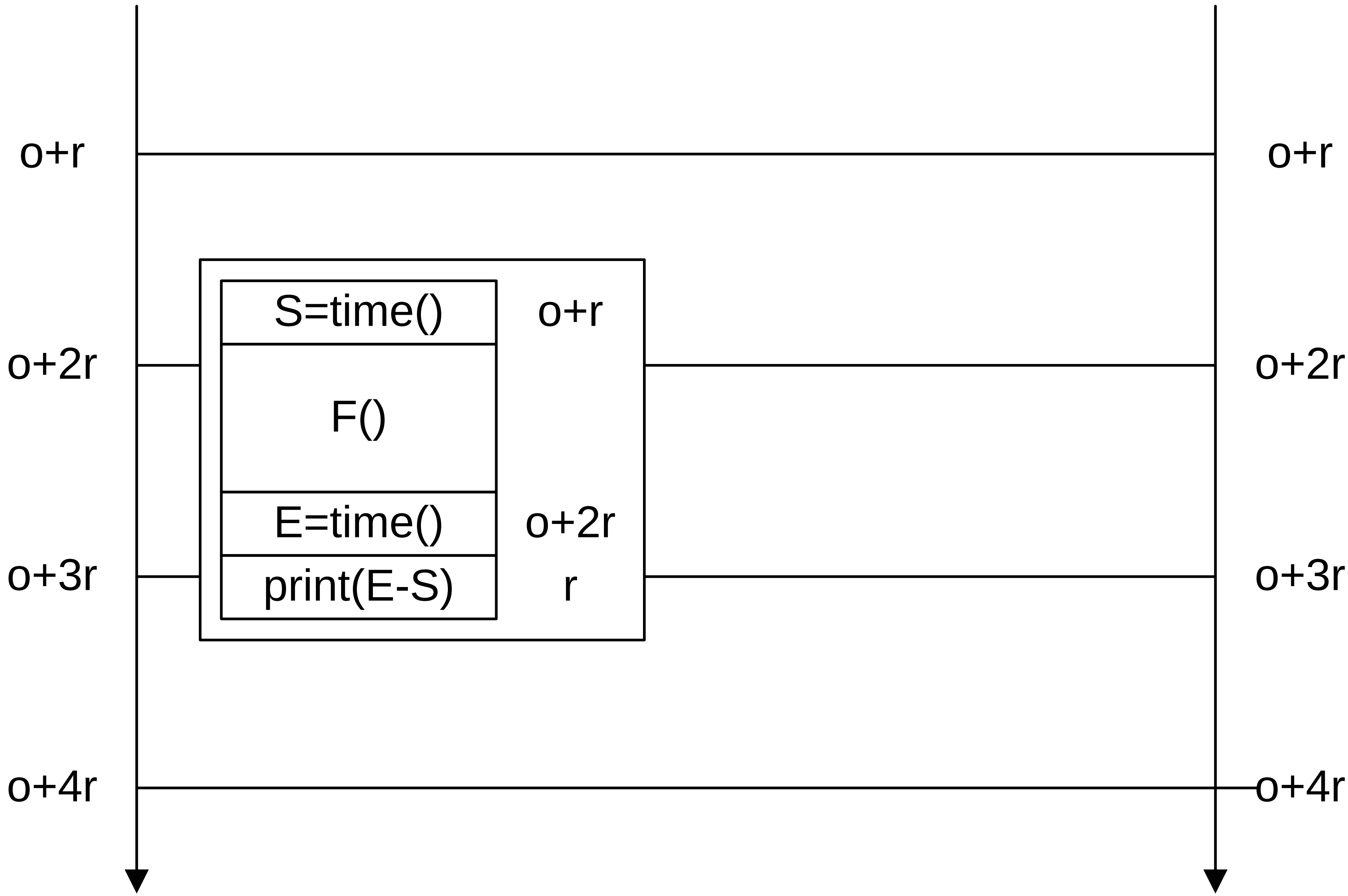
- How long does it take to complete some task X?
  - Fundamental property of a compute system
- Scale is elapsed time: seconds, hours, years(!)
  - Ratio scale: *can't take less than zero seconds*
  - Ratio scale: *one second is 2x better than two seconds*
- Need to carefully define what the components mean
  - What precisely is task X?
  - When does timing start, when does it stop?
  - What sort of time is being measured
    - Wall-clock time: e.g. human with a stop-watch
    - CPU time: only measure time when the task was executing

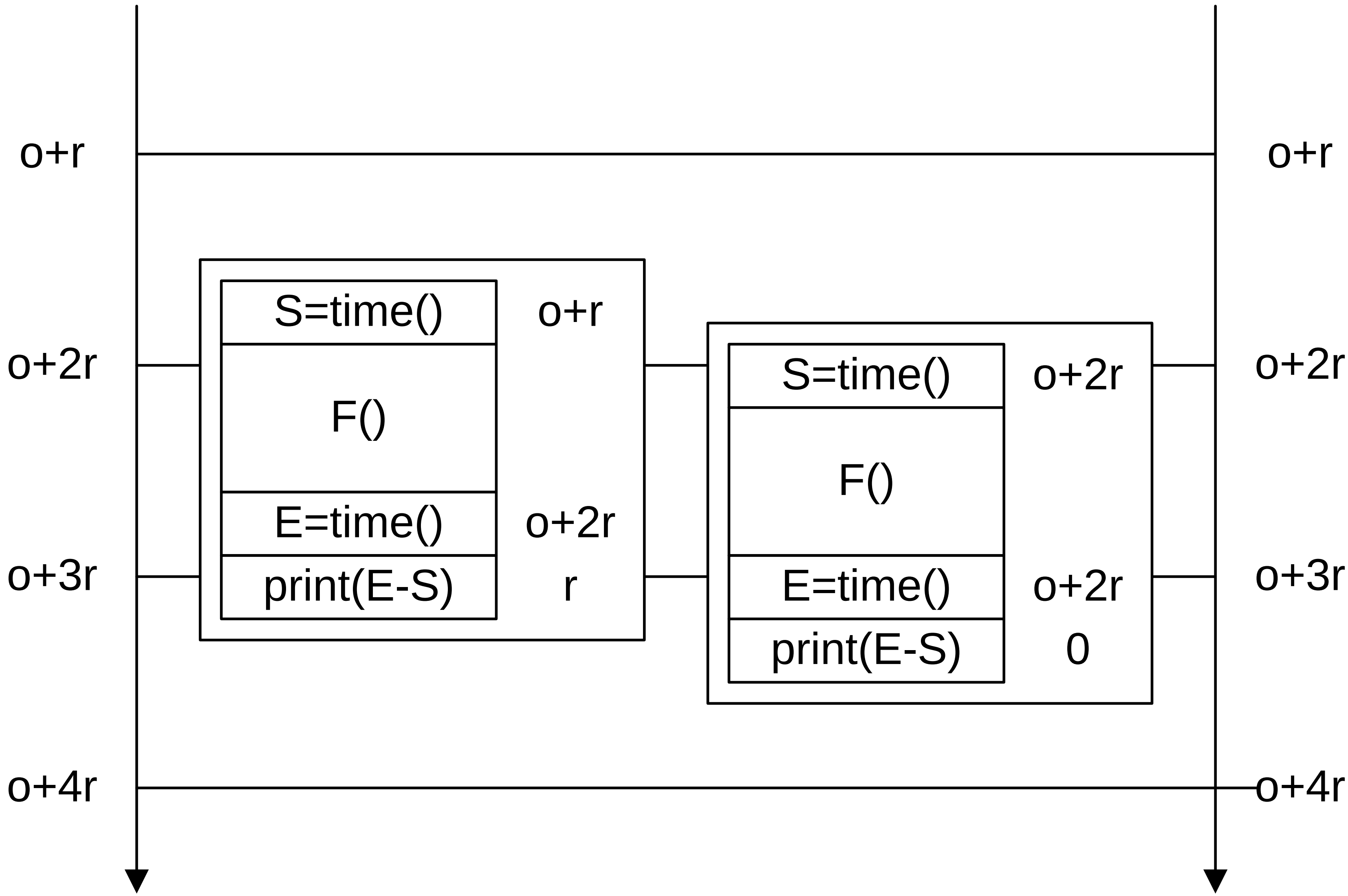
Measuring elapsed time

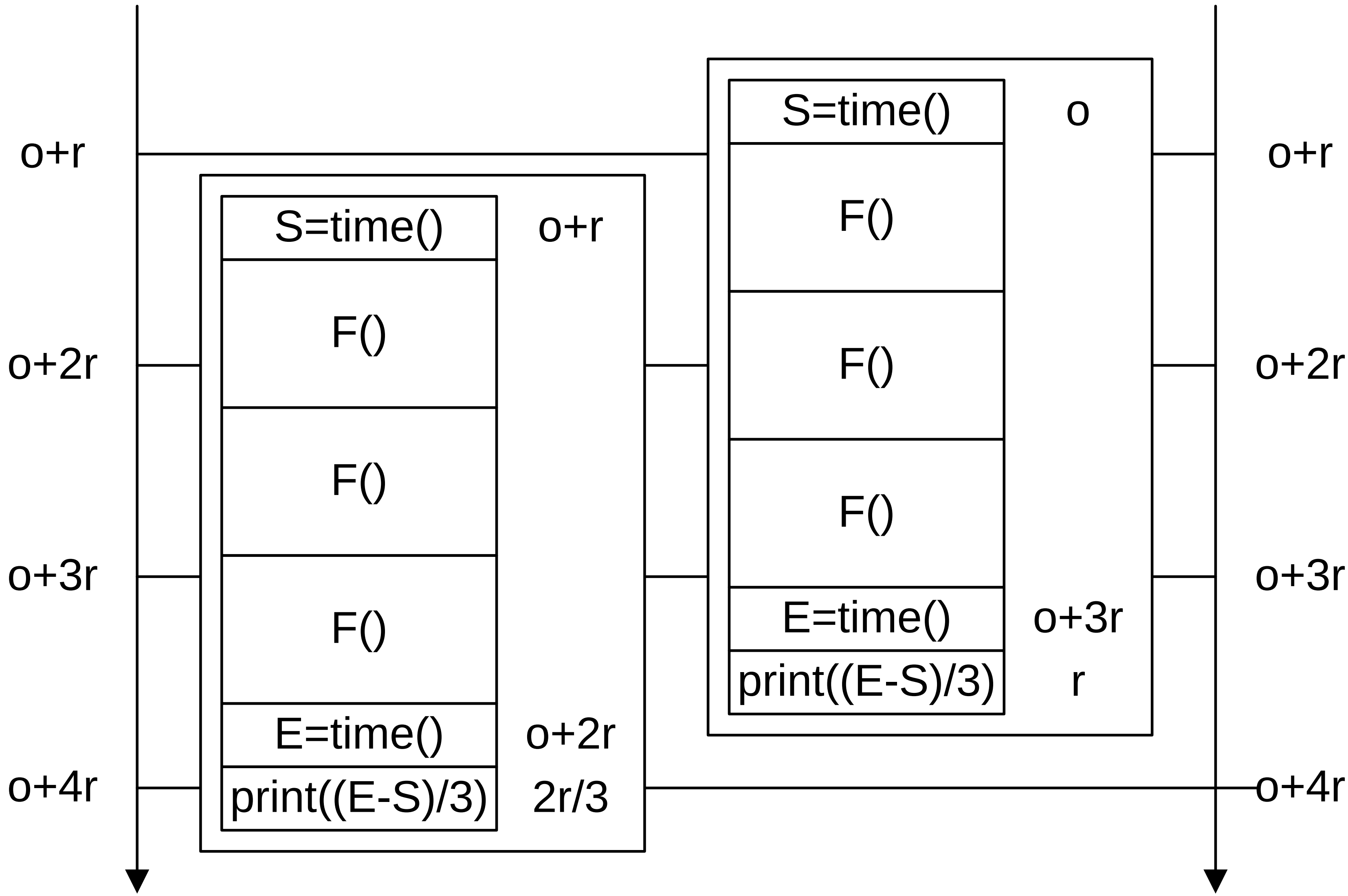
- This is often surprisingly difficult!
- We need some sort of timer as a base
  - **Precision**: Smallest representable time difference
  - **Resolution**: Smallest time difference you can measure
  - **Accuracy**: How reliable is the timer
- Timer can be part of the system, or external
  - External: you with a stop-watch, or pulse timer
  - Internal: the system itself records time during execution
- Internal timers are convenient, but tricky
  - Performing timing may change the systems performance!
  - Timer may not be consistent across the system

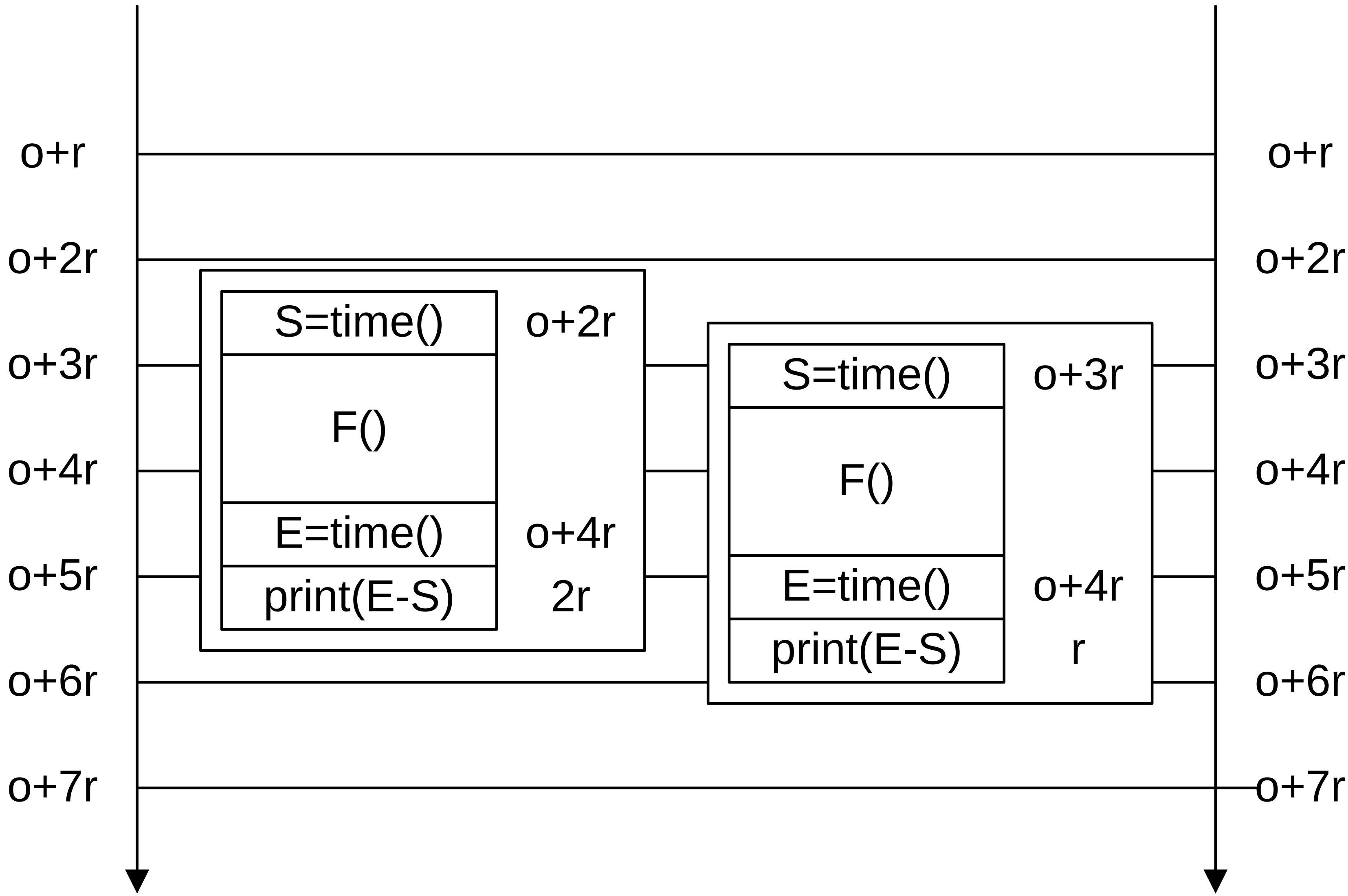

Time
F()
Execution Time


S=time()
F()
E=time()
print(E-S)


o+r
o+r
S=time()
o+r
o+2r
o+2r
F()
E=time()
o+2r
o+3r
print(E-S)
r
o+3r
o+4r
o+4r


o+r
o+r
o+2r
o+2r
o+3r
o+3r
o+4r
o+4r
S=time()
o+r
F()
E=time()
o+2r
print(E-S)
r
S=time()
o+2r
F()
E=time()
o+2r
print(E-S)
0


o+r
o+2r
o+3r
o+4r
S=time()
o+r
F()
F()
F()
E=time()
o+2r
print((E-S)/3)
2r/3
S=time()
o
F()
F()
F()
E=time()
o+3r
print((E-S)/3)
r
o+r
o+2r
o+3r
o+4r


o+r
o+r
o+2r
o+2r
S=time()
o+2r
o+3r
F()
o+3r
S=time()
o+3r
o+4r
o+4r
F()
E=time()
o+4r
o+5r
print(E-S)
2r
E=time()
o+4r
o+5r
print(E-S)
r
o+6r
o+6r
o+7r
o+7r

Internal timers

- Most platforms provide calls which will return elapsed time

	Windows	**Linux**	**x86 CPU**
Timer	QueryPerformanceCounter()	clock()	RDTSC
Resolution	micro-second	milli-second	nano-second
Measures	Wall-Clock	CPU-Time	Clock-Cycles
Consistency	Entire System	Current Thread	Current CPU

Problems when measuring time

- Start-up overhead: the first execution usually takes longer
  - Filling caches; compiling code; performing initialisation
  - **Cold-start**: restart system; time first execution of task
  - **Warm-start**: execute task once; measure the second execution
- Interference: other tasks are running on the system
  - Multi-user system: other users may execute system
  - Operating-system tasks: e.g. disk utilities affect performance
- Intrinsic variability: computers are not very deterministic
  - Cache contents differ significantly on each run
  - Unpredictable scheduling of thread by system
  - Interaction between resolution of timer and length of task
- General policy: measure over multiple executions and average

Throughput

- At what ***rate*** can task X be executed
  - Unit is executions per time
- Throughput is ***not*** just the inverse of elapsed time
  - Elapsed time is latency – time for a **single** task
  - Throughput is the execution rate for **many** tasks
- There is often a trade-off between latency and throughput
  - Parallel systems can do lots of tasks in parallel: good throughput
  - Difficult to split a single task into many pieces: poor latency

- Frames-per-second our obvious throughput example
  - Gaming: need high frames/second ***and*** low latency
  - Video: need high frames/second, but latency is not important

Energy

- Definitions of energy consumption are application specific
  - How much energy is used while executing task X?
  - What is the average power of a system capable of task X?
  - What is the peak power of a system executing task X?
- Closely related to temperature metrics
  - How hot does the device get while doing X?
  - What are the cooling requirements for continually doing X?
- Computers ***must*** consume energy: Landauer's principle
  - Any bit-wise operation (e.g. `and`, `or`) requires k*T*log(2) joules
    - k: Boltzmann constant; T: circuit temperature
  - e.g. a task requires 2^{64} bit-wise operations at 60 degrees C
    - $3.2*10^{-21}$ joules/bit -> ~0.05 joules
  - We're not that close to the limit yet...

Memory constraints

- All levels of storage: memory, cache, disk
  - All the working memory needed during calculation
  - May include input and output storage: are they the same place?
  - Memory constraints are balanced against communication costs
- There is an explicit size/bandwidth constraint

Level	Capacity	Bandwidth
Remote “Disk” (SAN)	10 EB	100 MB/sec
Local “Disk” (SSD)	10 TB	500 MB/sec
Memory (GDDR)	8 GB	300 GB/sec
Cache (local mem)	128 KB	2 TB/sec

- Often we will trade-off computation versus storage
  - Or programmer effort versus IO

Quality

- There is often a space of answers: how good is any given one?
  - Compression quality
  - Signal to noise ratio
- Qualitative quality metrics can often be made quantitative
  - User-interface responsiveness -> Input to display latency
  - Image quality -> root-mean-square-error against reference model
- If quality is a ratio measure, then there exists a “perfect” answer
  - But: realities of floating-point maths means this may not happen
- Understanding quality is critical for high performance
  - Achieving the “perfect” answer is often computationally intractable
  - Usually a trade-off between quality and speed
  - Need to choose the best tradeoff between the two

Making sure metrics are meaningful

- Some things are quantifiable, but not very useful
  - CPU performance: MHz is not the same as performance
  - Cameras: Mega-Pixels is not the same as quality
- Consistent and quantifiable metrics provide open competition
  - Suppliers of systems always want to use the “best” metrics
  - Metrics should be defined by users, not suppliers
- People will optimise for metrics (it’s what they are for!)
  - Poor metrics lead to poor design and optimisation
  - Part of the specification problem