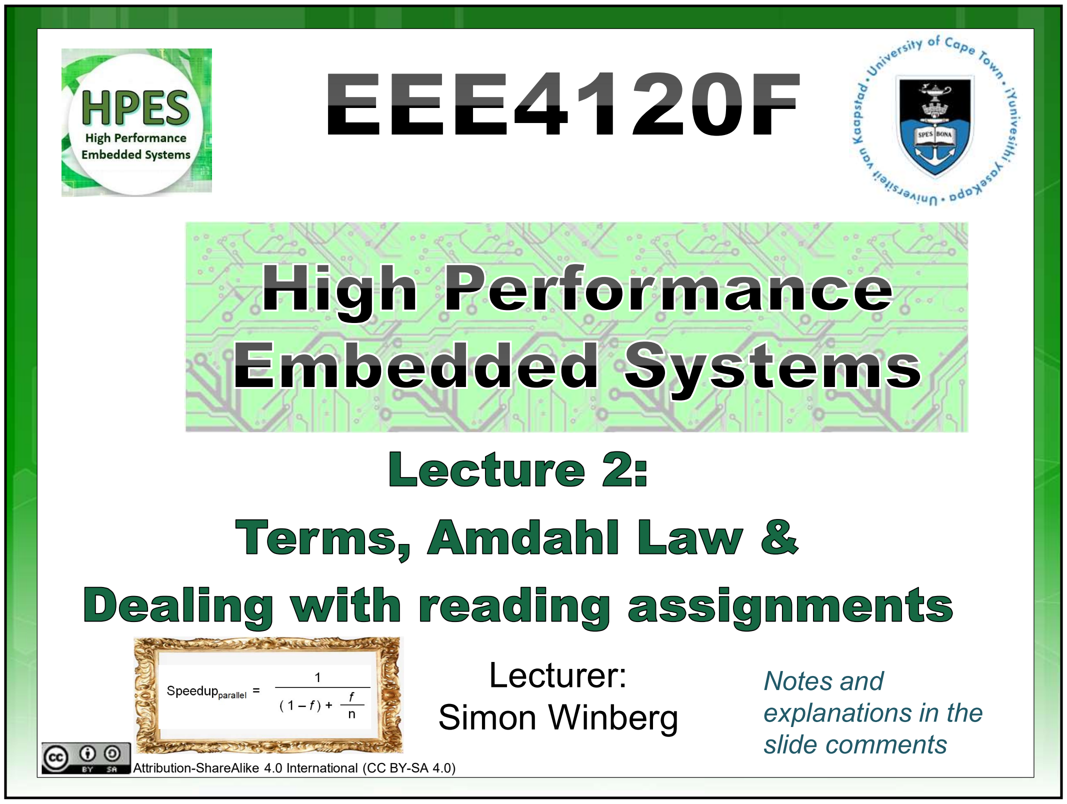


EEE4120F

High Performance Embedded Systems

Lecture 2: Terms, Amdahl Law & Dealing with reading assignments

$$\text{Speedup}_{\text{parallel}} = \frac{1}{(1-f) + \frac{f}{n}}$$

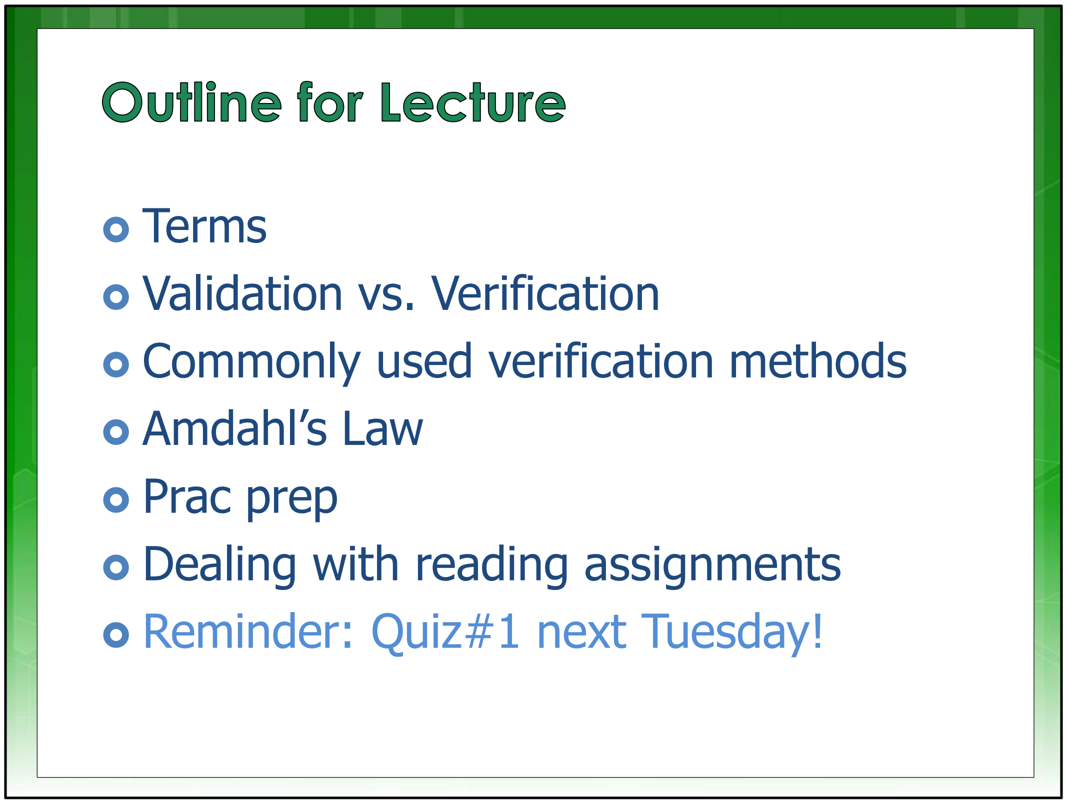
Lecturer:
Simon Winberg

Notes and explanations in the slide comments

The comments provide annotations to elaborate the slides, and can be used as study notes also.

You can use View -> Notes Page to view these comments below the slide for each slide. To generate pdf notes with slides, you just go to File -> Print and then “Change Full Page Slides” to “Notes Pages”.

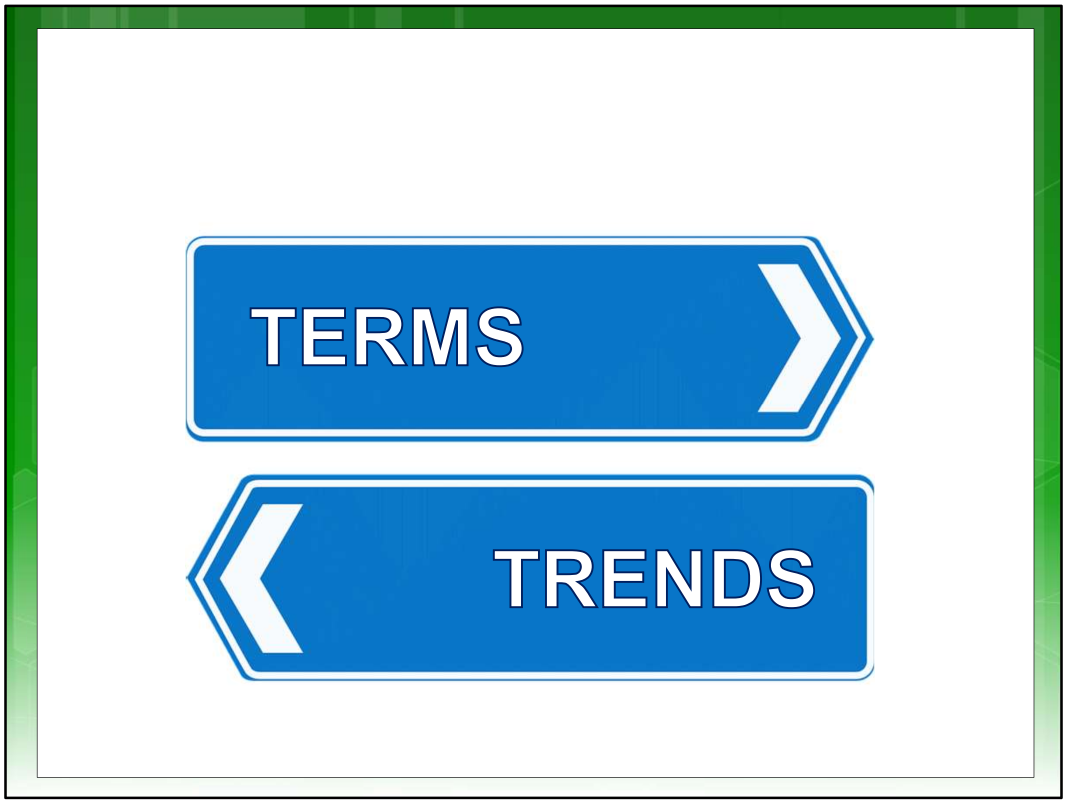
Comments for this slide1:
Hi there. In this lecture we will be getting into a few essential terms that we will use frequently in pracs and reviewing performance, and we get into the concept of **Amdahl's law**.



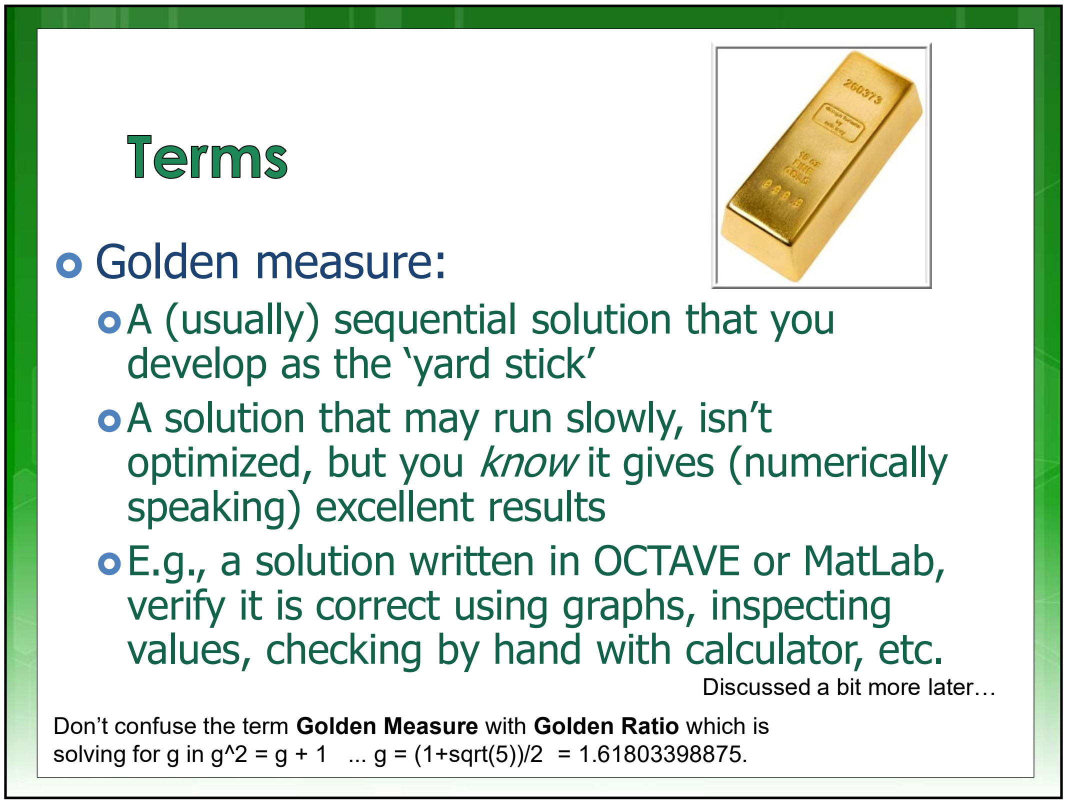
Outline for Lecture

- Terms
- Validation vs. Verification
- Commonly used verification methods
- Amdahl’s Law
- Prac prep
- Dealing with reading assignments
- Reminder: Quiz#1 next Tuesday!

Here's the outline, let's move on.



Let’s get in to essential Terms… and leave the more cool topics and thinking about fancy processors for a moment.



Terms

- Golden measure:
  - A (usually) sequential solution that you develop as the 'yard stick'
  - A solution that may run slowly, isn't optimized, but you *know* it gives (numerically speaking) excellent results
  - E.g., a solution written in OCTAVE or MatLab, verify it is correct using graphs, inspecting values, checking by hand with calculator, etc.

Discussed a bit more later…

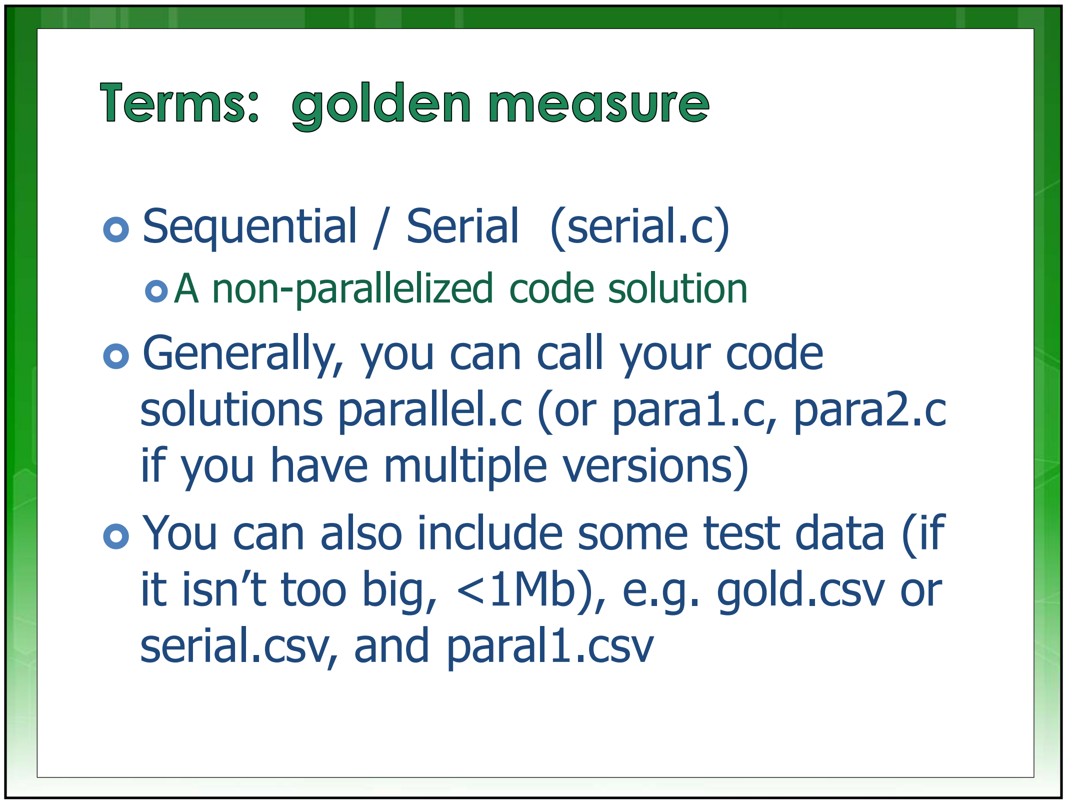
Don't confuse the term **Golden Measure** with **Golden Ratio** which is solving for g in g^2 = g + 1 ... g = (1+sqrt(5))/2 = 1.61803398875.

The term of "Golden Measure" will be used often in this course. It sounds nice and shiny, a good start. But actually, the term Golden Measure is used in computing more to refer to a solution that produces accurate results but doesn't necessarily run fast or efficiently. It is commonly used as a baseline for comparison to or between other computing solutions that solve the same task. So Golden here is really an adjective meaning accurate ... ironically these Golden Measures are usually way much cheaper to construct than the, probably less perfectly correct, optimized parallel solutions being compared against.

Please don't confuse the term **Golden Measure** with **Golden Ratio** which is solving for g in g^2 = g + 1 ... g = (1+sqrt(5))/2 = 1.61803398875. That 1.618 is the golden ratio which is perhaps useful for some engineering work.

While on the topic of golden radio, if you excuse getting side-tracked, you might like to try the Fibonacci golden ratio algorithm… which could be done as a YODA topic ;)

Golden measure itself is not necessarily a universally defined and used term for a yardstick of computation, but the term does come up in some texts and academic papers.



Terms: golden measure

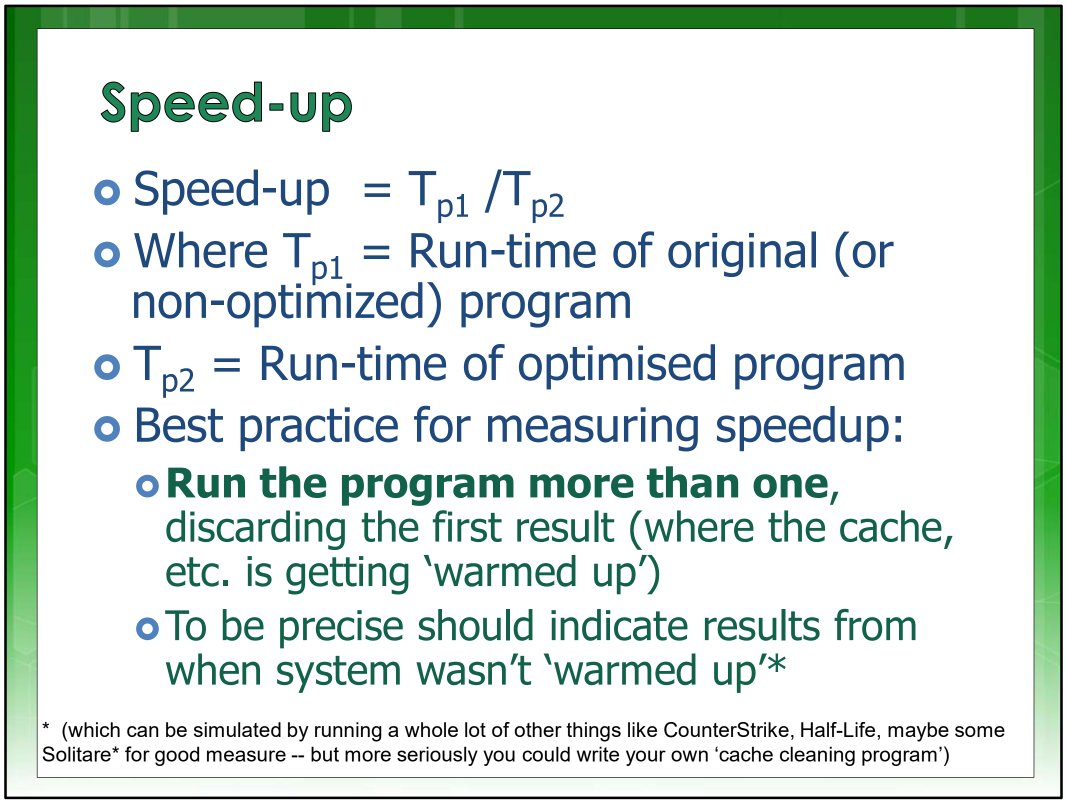
- Sequential / Serial (serial.c)
  - A non-parallelized code solution
- Generally, you can call your code solutions parallel.c (or para1.c, para2.c if you have multiple versions)
- You can also include some test data (if it isn't too big, <1Mb), e.g. gold.csv or serial.csv, and paral1.csv

In some of the pracs you will be given a starting point of non-parallel code, usually called serial dot C (sorry, I was being funny, usually called "serial.c" if one is typing). Then you can call you parallelized code something like "parallel.C" to make it easy for the marker to find.

You should also include some test data files.

An example file name is given here, in this slide.

If you are given a big file as part of the prac assignment please do not submit that file as we would have it already.



Speed-up

- Speed-up = T_{p1} / T_{p2}
- Where T_{p1} = Run-time of original (or non-optimized) program
- T_{p2} = Run-time of optimised program
- Best practice for measuring speedup:
  - **Run the program more than one**, discarding the first result (where the cache, etc. is getting 'warmed up')
  - To be precise should indicate results from when system wasn't 'warmed up'*

* (which can be simulated by running a whole lot of other things like CounterStrike, Half-Life, maybe some Solitare* for good measure -- but more seriously you could write your own 'cache cleaning program')

The concept of speed-up is a measure of how much faster one program, call it P1, is compared to another, call it P2. The equation is, obviously, just time of P1 over time of P2. Usually the numerator, i.e. P1, is the non-optimized program or Golden Measure, and P2 is the optimized program. So if Golden Measure P1 takes 2 seconds and P2 takes 1 second, what is the speed up? It is 2 over 1, i.e. 2. Speed up has no units, although if you really need a unit you could use times. My P2 has a 2 Times speedup.

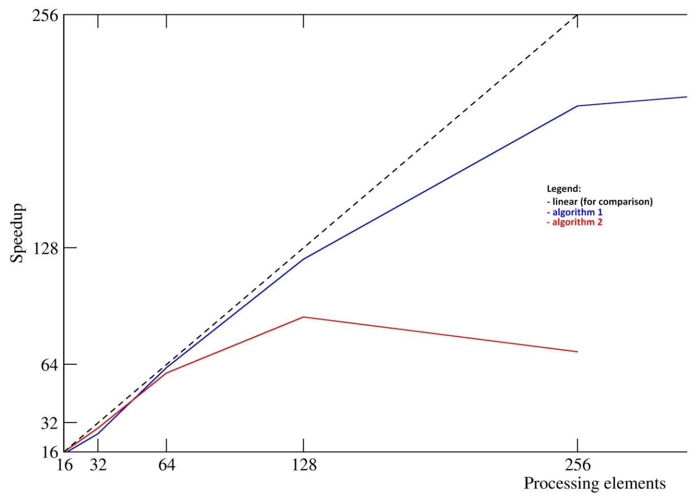
(can skip:)
As per definition on Wikipedia:
In computer architecture, **speedup** is a number that measures the relative performance of two systems processing the same problem. More technically, it is the improvement in speed of execution of a task executed on two similar architectures with different resources. The notion of speedup was established by Amdahl's law, which was particularly focused on parallel processing. However, speedup can be used more generally to show the effect on performance after any resource enhancement.

source: https://en.wikipedia.org/wiki/Speedup

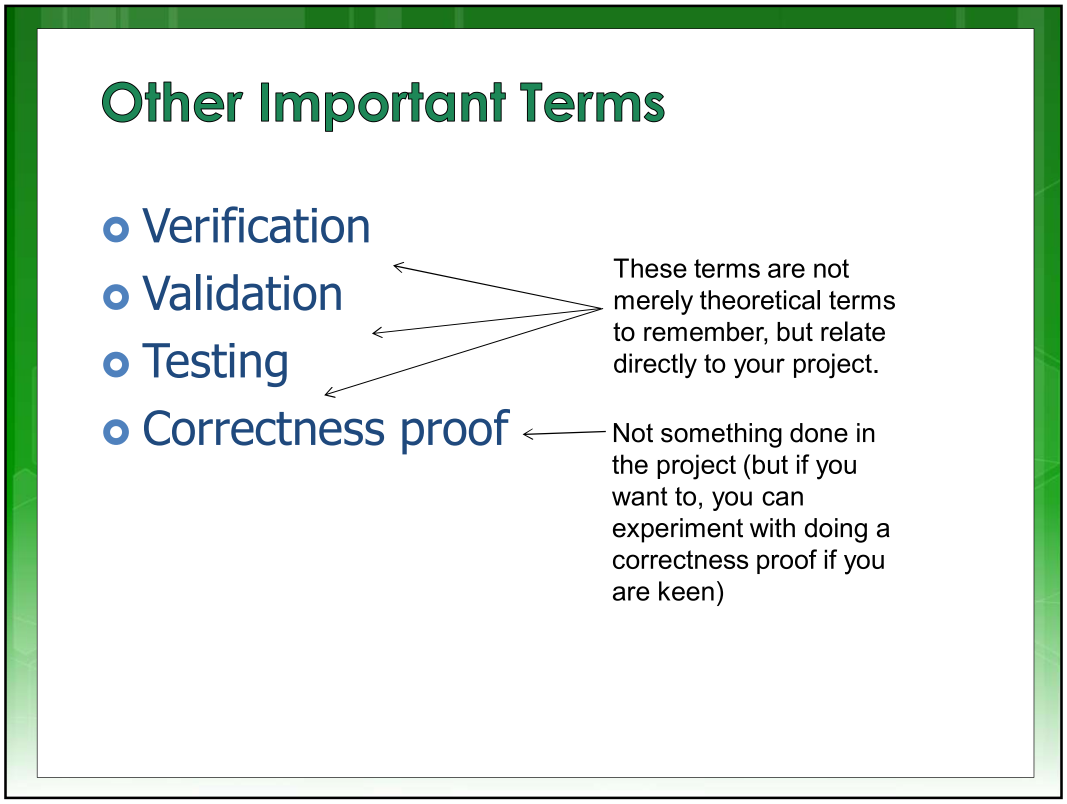
Discussion about the definition of speed-up:

Speedup is a number for expressing the relative performance of two systems processing the same problem or completing the same processing solution (either 'same problem' or 'same processing solution' would be considered correct, although generally, and I know it's a somewhat philosophical view, it would be better saying the same problem or solution, for which we want to get similarly correct results out, but we don't mind what the specific processing is as long as it gives correct (or sufficiently accurate) solutions; for example sequential and parallel processing implementations that solve a given problem would likely be different, thus it is more correct to "say solving the same problem" as apposed to "doing the same processing".

Speed-up graphs



This is a typical speed up graph, which one sees in many computing type papers. The vertical axis is Speedup. The horizontal axis is often number of cores. But! The horizontal is actually, more generally, a metric that rates the characteristic, usually hardware size, of the architectures you are contrasting. So you could perhaps have an 8 bit Pick architecture on the left, and a 64 bit I 7 on the right and compare how quickly they do a one oh two four floating point F F Tee. You would probably find that the 64 bit gives a hefty speedup.



Other Important Terms

- Verification
- Validation
- Testing
- Correctness proof

These terms are not merely theoretical terms to remember, but relate directly to your project.

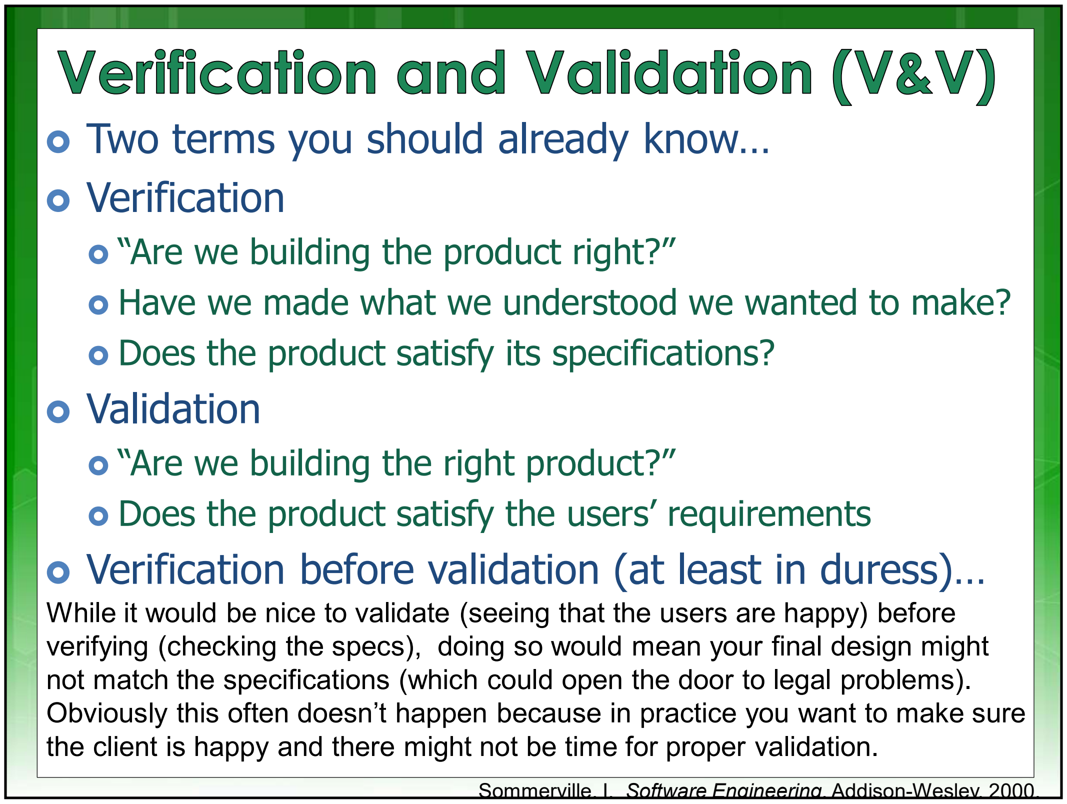
Not something done in the project (but if you want to, you can experiment with doing a correctness proof if you are keen)

Here are some other important terms.

Actually, these terms that are commonly used in engineering in relation to product or prototype development. You should really know what is meant by Verification, Validation and Testing of a product.

In this course these are used in the Conceptual Assignment this term and in the YODA project next term.

The theory of Correctness Proofs is a wonderful thing. But unfortunately it is not within the scope of this course. Although, it would certainly make for a good Masters level course.


Verification and Validation (V&V)

- Two terms you should already know…
- Verification
  - “Are we building the product right?”
  - Have we made what we understood we wanted to make?
  - Does the product satisfy its specifications?
- Validation
  - “Are we building the right product?”
  - Does the product satisfy the users’ requirements
- Verification before validation (at least in duress)…

While it would be nice to validate (seeing that the users are happy) before verifying (checking the specs), doing so would mean your final design might not match the specifications (which could open the door to legal problems). Obviously this often doesn’t happen because in practice you want to make sure the client is happy and there might not be time for proper validation.

Sommerville, I., *Software Engineering*, Addison-Wesley, 2000.

So, you might be wondering, what is Verification and Validation?

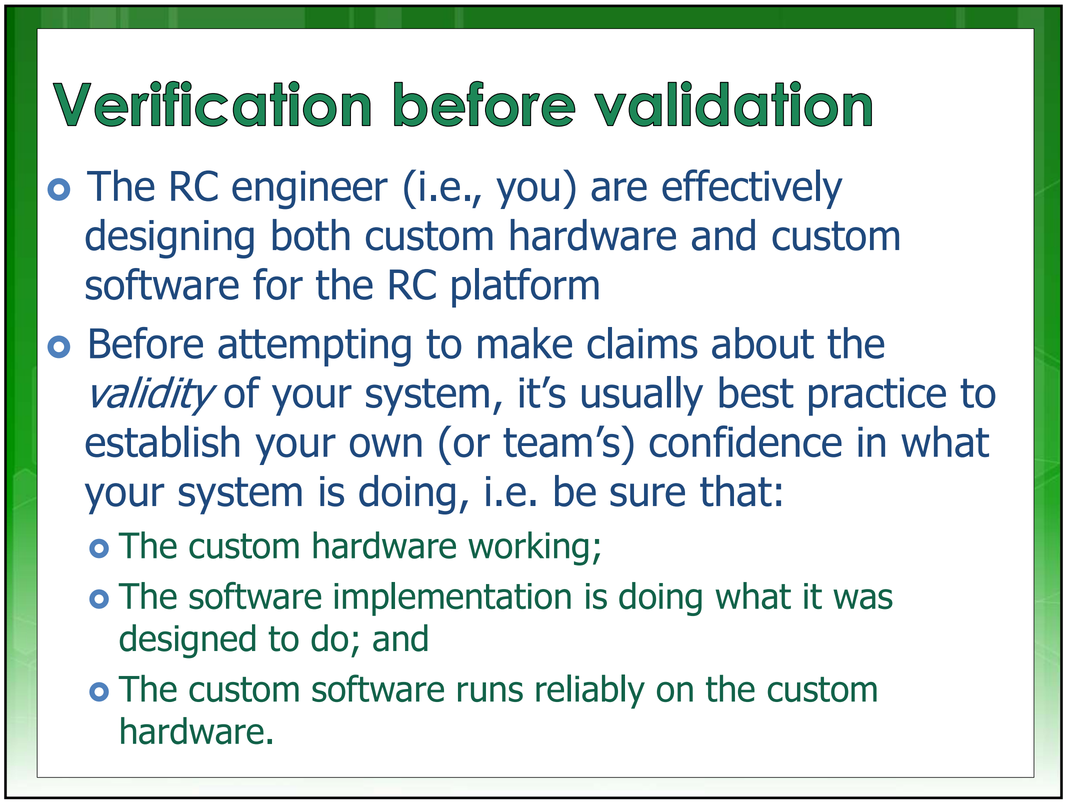
Verification, which you might have heard in the design course, is checking: are we building the product, right. Note that it not checking if we are building the right, product. It is confirming that we are building the product we said we would build, according to the specifications and contract with the client. We aren't checking here if the client is delighted with the product we deliver.

Now Validation, on the other hand is checking if we have built the right product. Yes, this is where you call the client in. Maybe some of their employees too. And get them to try the system, if possible, in situ. Where it would be used. Oh, and don’t forget this would involve some good coffee and cakes, perhaps taking the happy clients to lunch. Oops, sorry that went off topic. Basically, validation is checking if user needs are met not just that agreed specifications are achieved.

(additional point and clarity on specifications for product validation:)

Remember that for industry products, the specifications is usually a document that the client agrees to and signs off on, it is technically a legally binding contract.

So, if you are thinking short-term bottom line and business survival, then you basically want to ensure you are getting the design right according to the agreed upon specifications. But perhaps if you are thinking long-term survival, then you probably want to ensure clients are happy and this may mean a tradeoff between what specs are satisfied and how satisfied the client is. Usually there is some flexibility that the client may agree to e.g. modify certain specs in order to get the product working sooner.



Verification before validation

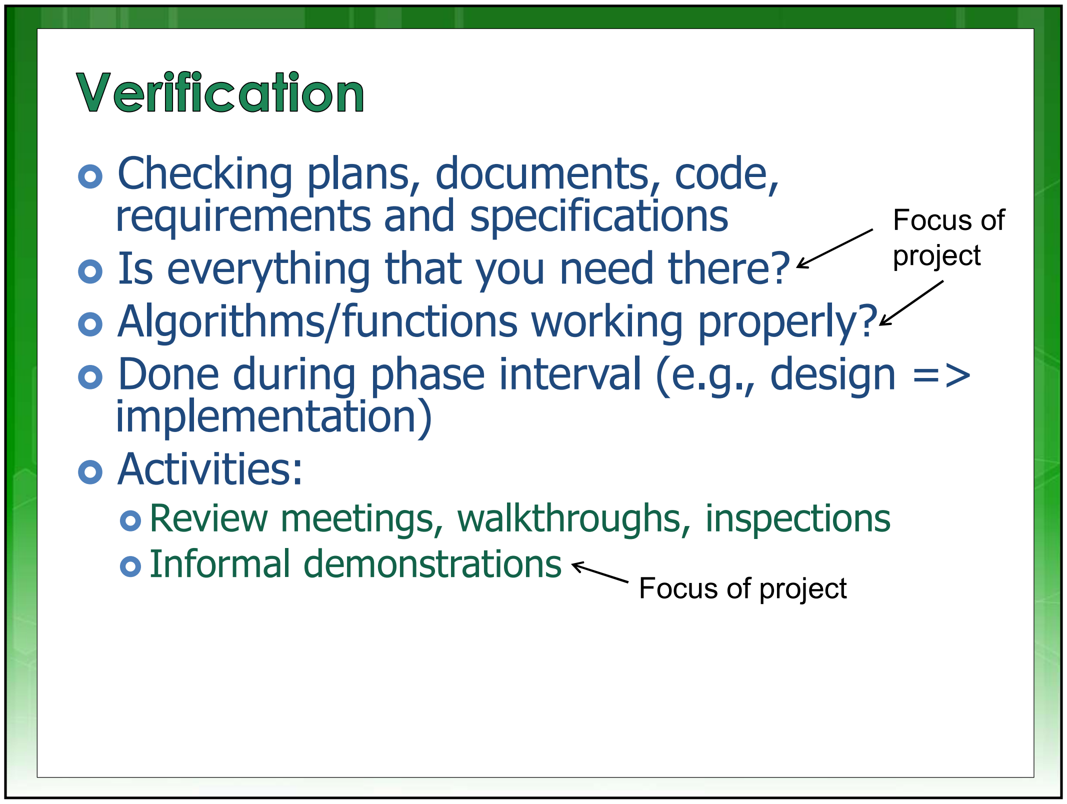
- The RC engineer (i.e., you) are effectively designing both custom hardware and custom software for the RC platform
- Before attempting to make claims about the *validity* of your system, it's usually best practice to establish your own (or team's) confidence in what your system is doing, i.e. be sure that:
  - The custom hardware working;
  - The software implementation is doing what it was designed to do; and
  - The custom software runs reliably on the custom hardware.

You might have noticed my additional note in the previous slide about often validation before verification. Or being nice to the client. However a the RC team. More explicitly, the Reconfigurable Computing Engineering team, usually wants to do verification before validation. Such engineers typically build custom hardware and custom software. So, before making claims about the validity of your entire system it is typically better for the team to do verification first. For example to confirm that the interfaces are correct, and sub-modules work as planned. This is to ensure the team is happy with the parts of the system. And that the parts adhere to specifications. Then only is validation of the system done. In the validation it might be found that requirements are not met so the team might have to do some redesign. But this method has advantages. Such as avoiding individual team members being blamed for not having adhered to specifications.

I like Henry Ford's quote on teamwork, which is a strong inspiration and motivation for the importance of teams in development. It goes like this:

> "Coming together is a beginning. Keeping together is progress. Working together is success."

A good team leader needs to think how to keep the team working well together.



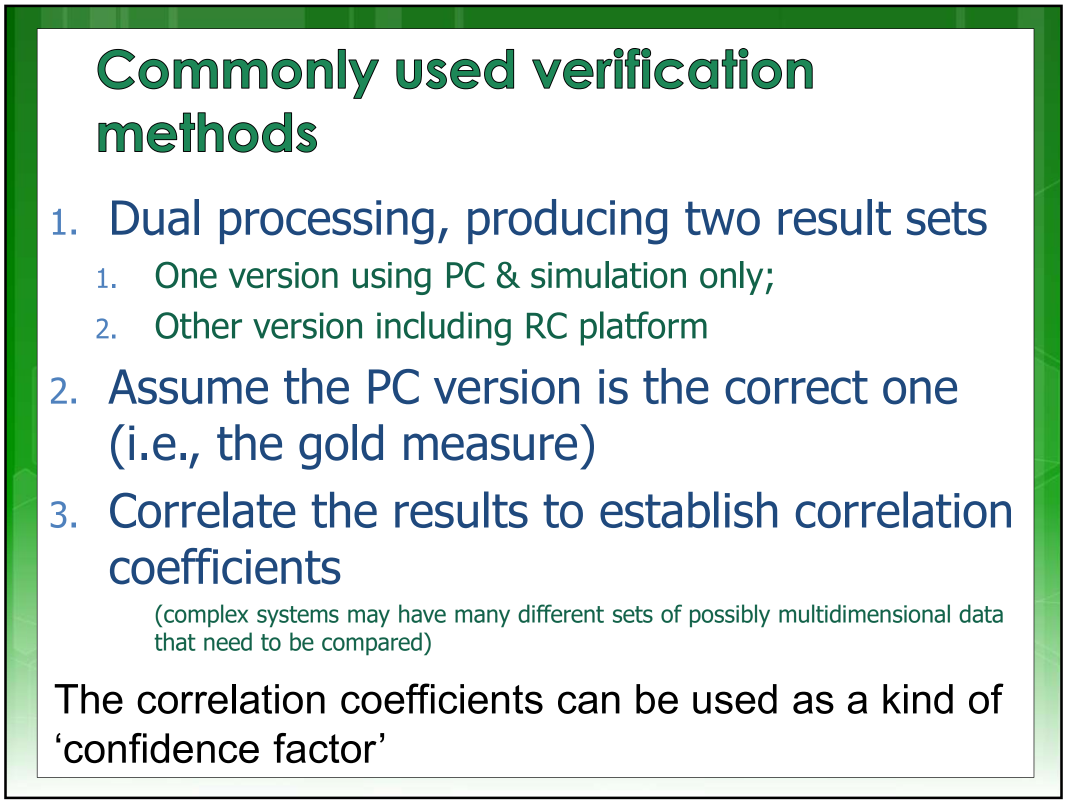
Verification

- Checking plans, documents, code, requirements and specifications
- Is everything that you need there?
- Algorithms/functions working properly?
- Done during phase interval (e.g., design => implementation)
- Activities:
  - Review meetings, walkthroughs, inspections
  - Informal demonstrations

Focus of project

Focus of project

The activities of verification include: Review meetings, walkthroughs, inspections, and Informal demonstrations. We plan to have some of these in the YODA project.


Commonly used verification methods

1. Dual processing, producing two result sets
   1. One version using PC & simulation only;
   2. Other version including RC platform
2. Assume the PC version is the correct one (i.e., the gold measure)
3. Correlate the results to establish correlation coefficients

   (complex systems may have many different sets of possibly multidimensional data that need to be compared)

The correlation coefficients can be used as a kind of ‘confidence factor’

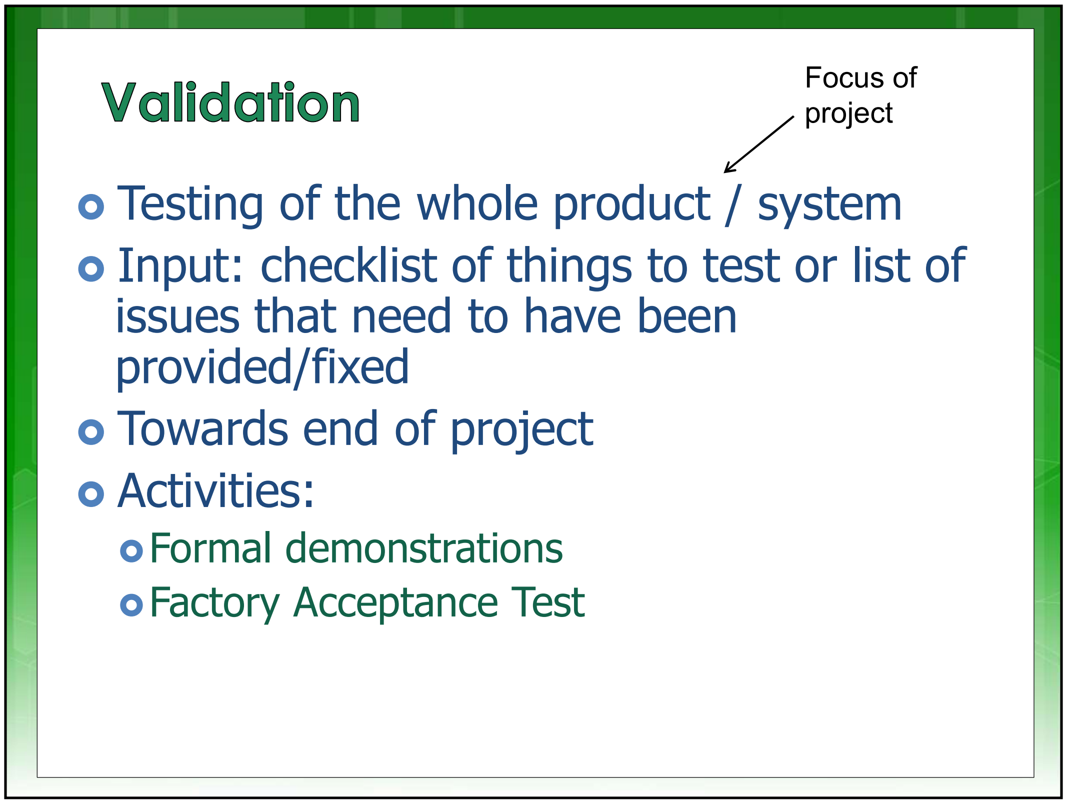
A commonly used verification method in HPES includes…

Dual processing. Which is about developing two different implementations that process the same result. This term, you might be fascinated to know, actually arises from psychology. In which it refers to how thought can arise in two different ways. From a conscious process, in which there is a clear logic towards the result. Or from an unconscious process, in which the process is not necessarily know and the result seems to just appear. Basically, the dual processing in computing that we use here is much the same. It still refers to the human though, the engineers building systems. Not to the machine (we are not thinking about Ais building machines at this point).

Let us consider a scenario of two systems. You put together a golden measure in MATLAB and also built an HPES system. They both produce the same result. The process of how the golden measure works, which is put together in MATLAB is quite similar to an unconscious process. We don’t necessarily know how it got from a short program written in a few minutes to a very accurate result. Then we have the other process, the HPES system. In which you’re likely worked from the bottom up. It may have taken a month. Or years to build. You consciously know every little step of the way of how your solution got from its initial, anguished booting. Through its teething and squeaking stage. Possible along the way it had to get turned on and off a bit and had other disciplining measure applied to make it behave. And it finally respond with a result.

So those are two processes. One that was perhaps quick and which you don’t really care much how it got to the result. And the other. Where you know exactly how it got to the result. And thusly you now hopefully have a clear understanding of what is meant by dual processing.

To simplifying things down though, one just looks at the results and not thinking so much about the process. And when it is about comparing results, a good and well-defined way of doing this is the use of correlation. In Prac 1 we make use of correlation to compare the accuracy of two computing methods.

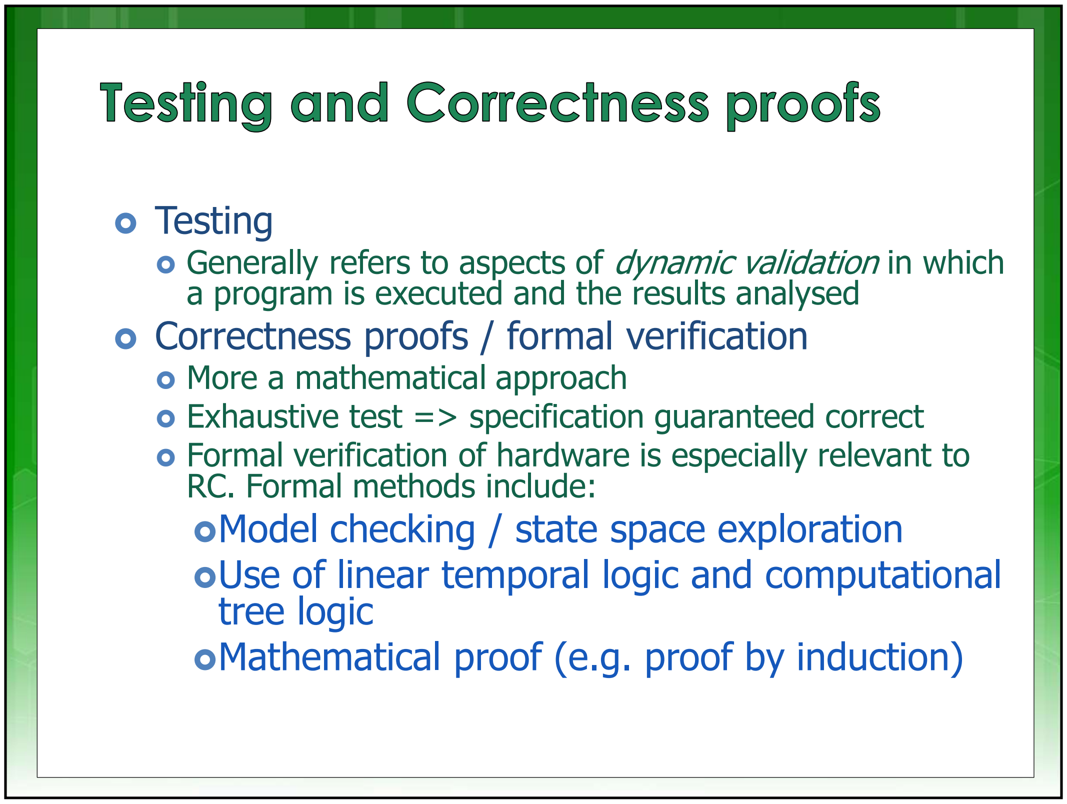


Validation

Focus of project

- Testing of the whole product / system
- Input: checklist of things to test or list of issues that need to have been provided/fixed
- Towards end of project
- Activities:
  - Formal demonstrations
  - Factory Acceptance Test

Validation is about testing of the whole product. The acceptance test is commonly a significant milestone in a project in which the product is validated to check that it is meeting the needs of the client, or more specifically that the system is satisfactorily achieving its purpose.

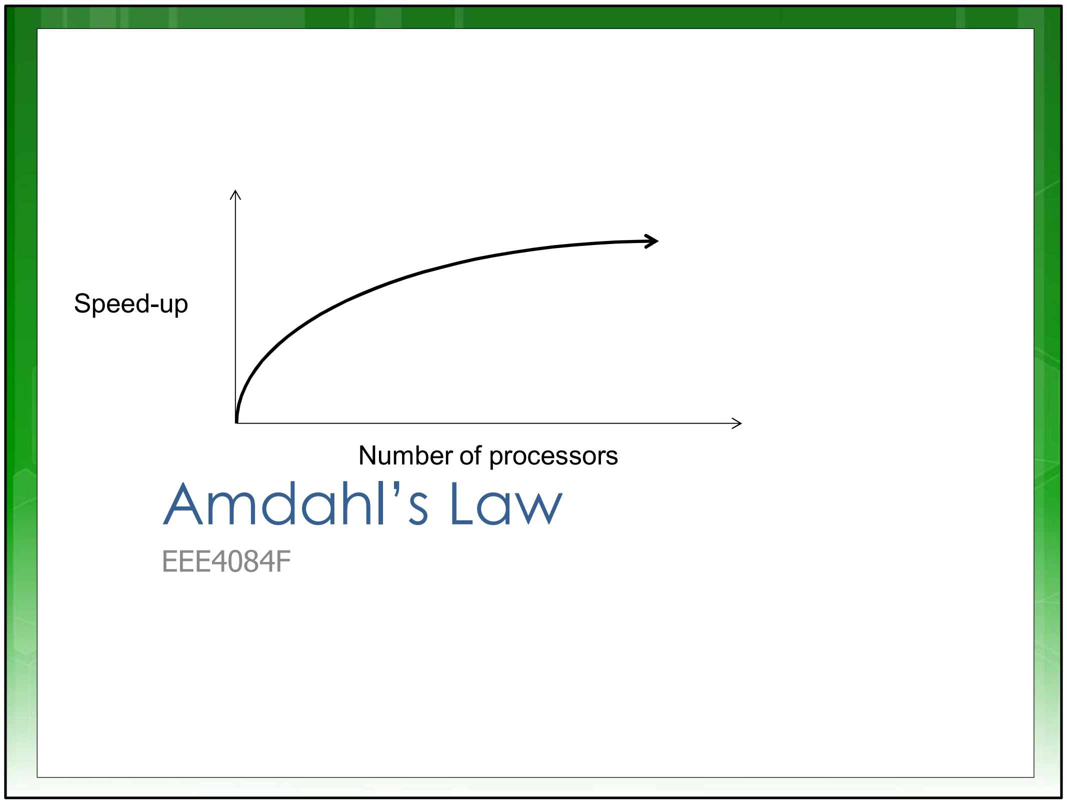


Testing and Correctness proofs

- Testing
  - Generally refers to aspects of *dynamic validation* in which a program is executed and the results analysed
- Correctness proofs / formal verification
  - More a mathematical approach
  - Exhaustive test => specification guaranteed correct
  - Formal verification of hardware is especially relevant to RC. Formal methods include:
    - Model checking / state space exploration
    - Use of linear temporal logic and computational tree logic
    - Mathematical proof (e.g. proof by induction)

Testing usually refers to aspects of dynamic validation, in which a program is executed and the results analysed. Acceptance Testing is mainly just a more structured and comprehensive testing activity.

This should not be confused with correctness proofs or formal verification. This is a more mathematical approach. Typically means exhaustive testing to mathematically (commonly using logic and inference rules) to prove specifications are guaranteed correct. One would do this in very high stakes, safety-critical systems like a rocket guidance control system.

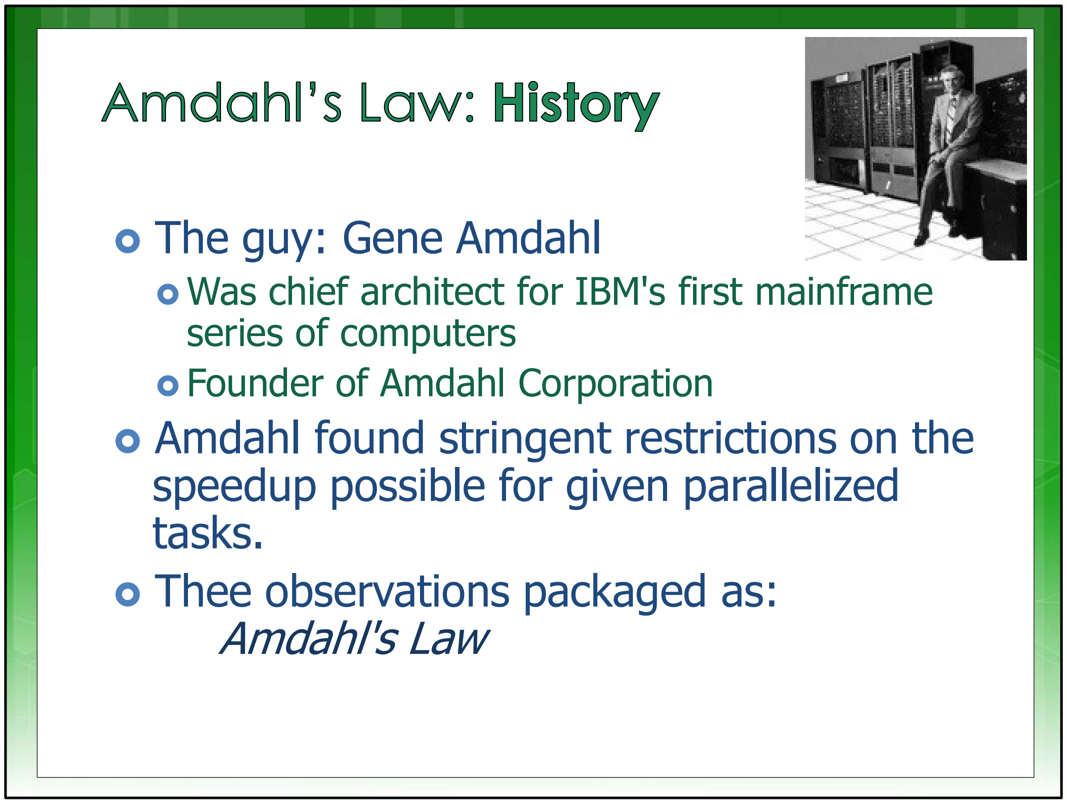


Amdahl's Law

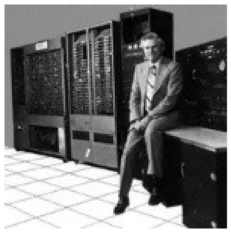
EEE4084F

We have now covered the important essentials of Speed up, Validation and Verification.

Now for Amdahl …

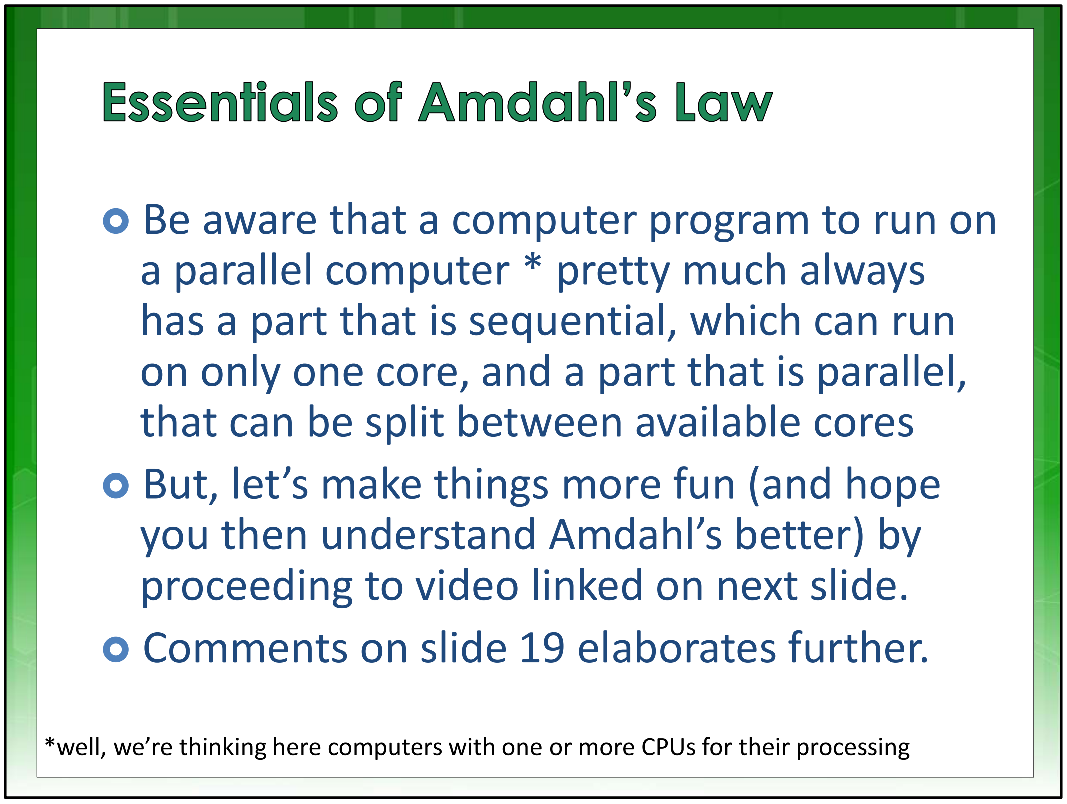


Amdahl's Law: **History**



- The guy: Gene Amdahl
  - Was chief architect for IBM's first mainframe series of computers
  - Founder of Amdahl Corporation
- Amdahl found stringent restrictions on the speedup possible for given parallelized tasks.
- Thee observations packaged as:
  Amdahl's Law

The main man here. Gene Amdahl. Was chief architect for IBM's first mainframe series. Amdahl found stringent restrictions on the speed-up possible for given parallelized tasks, and the essential ones are packaged as Amdahl's Law.

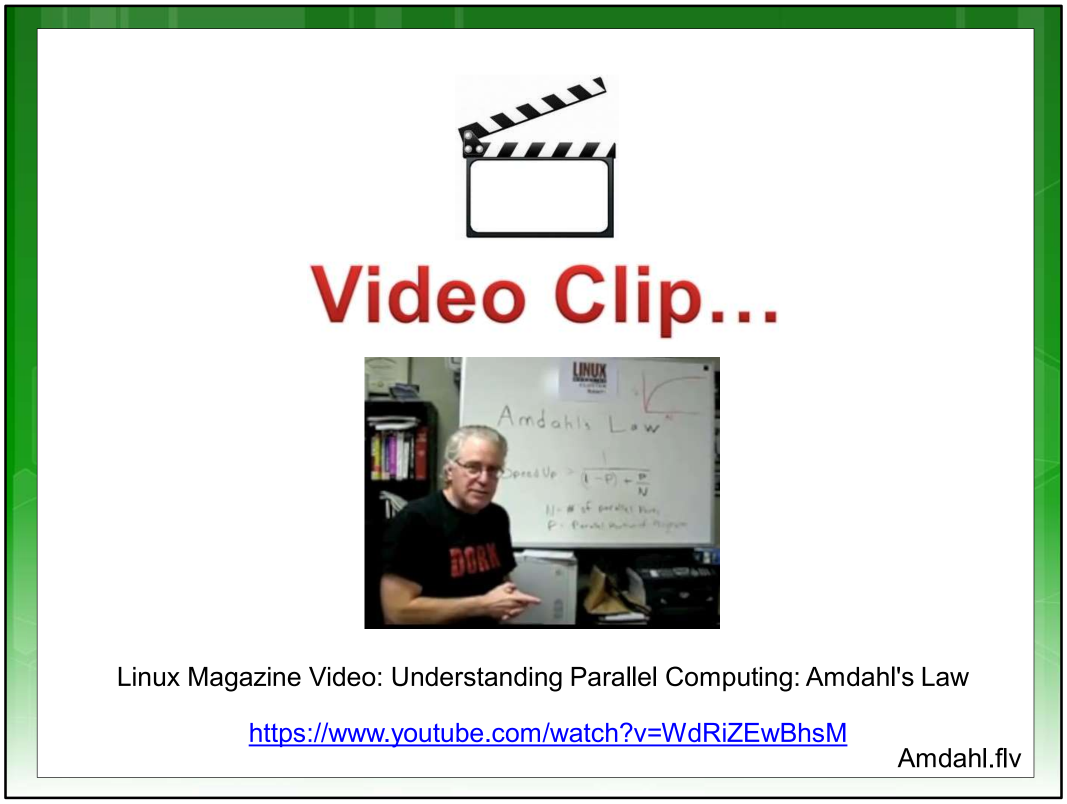


Essentials of Amdahl's Law

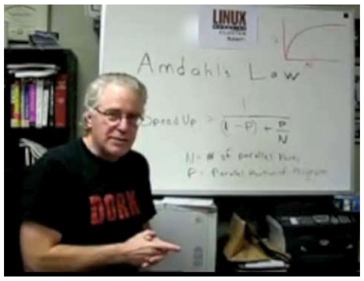
- Be aware that a computer program to run on a parallel computer * pretty much always has a part that is sequential, which can run on only one core, and a part that is parallel, that can be split between available cores
- But, let's make things more fun (and hope you then understand Amdahl's better) by proceeding to video linked on next slide.
- Comments on slide 19 elaborates further.

*well, we're thinking here computers with one or more CPUs for their processing

See comments on slide 19



Video Clip…

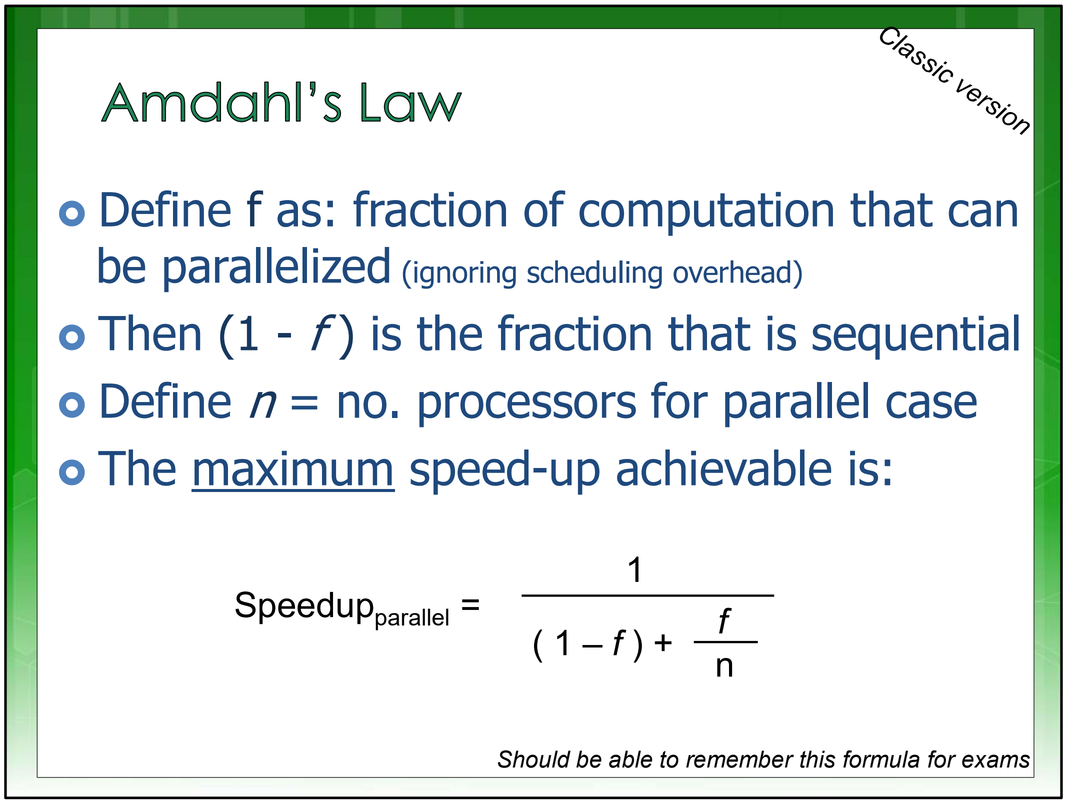

Amdahl's Law
Speed Up = 1 / ((1 − P) + P/N)
N = # of parallel Runs
P = Parallel Portion of Program

Linux Magazine Video: Understanding Parallel Computing: Amdahl's Law

https://www.youtube.com/watch?v=WdRiZEwBhsM

Amdahl.flv

Watch the video. Even though his T-Shirt logo might be discouraging, his wit and good explanation makes up for things.



Classic version

Amdahl's Law

- Define f as: fraction of computation that can be parallelized (ignoring scheduling overhead)
- Then (1 - *f*) is the fraction that is sequential
- Define *n* = no. processors for parallel case
- The <u>maximum</u> speed-up achievable is:

$$\text{Speedup}_{\text{parallel}} = \frac{1}{(1 - f) + \frac{f}{n}}$$

Should be able to remember this formula for exams

This is the essential Amdahl's law. It works as follows.

You should be aware that a computer program to run on a parallel computer pretty much always has a part that is sequential, which can run on only one core, and a part that is parallel, that can be split between available cores.

Yes, you may be aware this is a simplification. You could have a complicated beast of a HPES that has more than just multiple microprocessor cores, to accelerate tasks.

But put such complicated beasts out of you mind for now. Just think multiple cores. With only one core. The master core, which is often called C 0. As the core that gets the party started.

We can thus define things as follows:

F is defined as the fraction of computation that can be parallelized.

N is the number of processors for the parallel case.

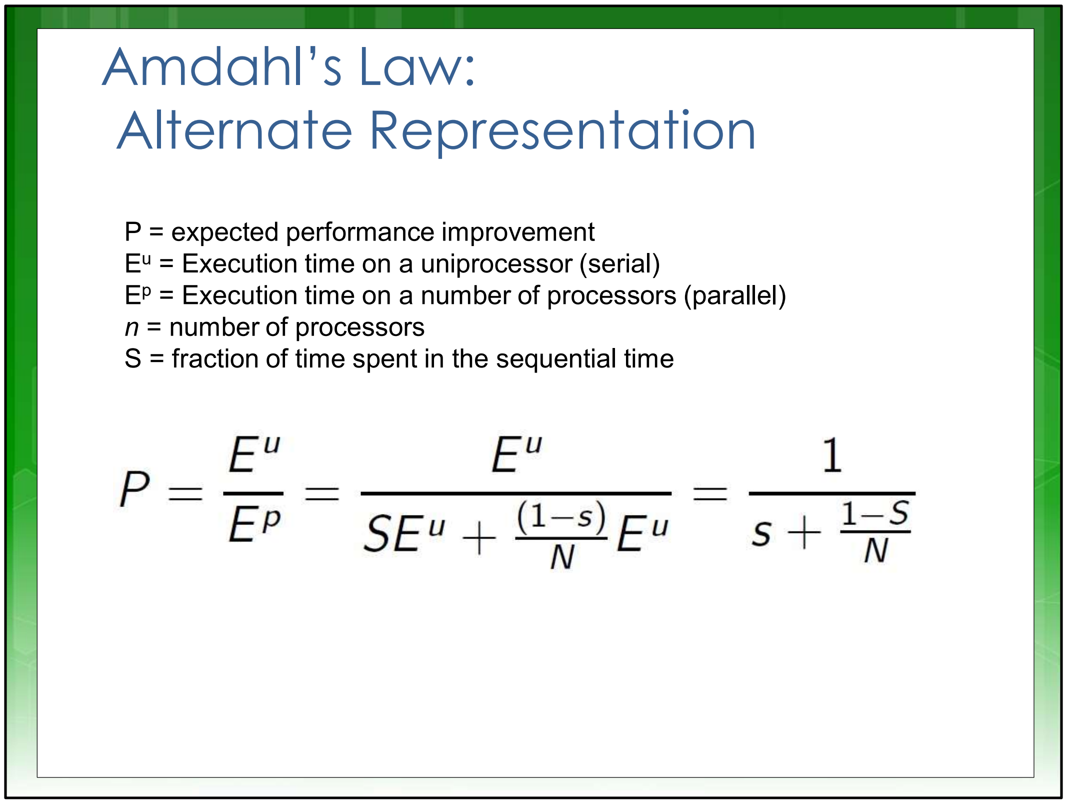
One minus F is thus the part that is sequential.

Using these parameters Amdahl's law is given by:

One upon the denominator one minus F plus F divided by N.

That is it. You can see it is tightly bound by only the parallelizable portion being able to be speeded up.

Typically, there are two points along the horizontal axis of a speed up graph. One on the left where the speed up of parallelized execution improves. And one where adding additional cores has diminishing returns. This is caused by either adding cores that do not much improve speedup. Or, even worst, when adding cores starts to slows things down because the computer is doing more work adding threads than is by having more threads work on the problem.



Amdahl's Law: Alternate Representation

P = expected performance improvement
E^u = Execution time on a uniprocessor (serial)
E^p = Execution time on a number of processors (parallel)
n = number of processors
S = fraction of time spent in the sequential time

$$P = \frac{E^u}{E^p} = \frac{E^u}{SE^u + \frac{(1-s)}{N}E^u} = \frac{1}{s + \frac{1-S}{N}}$$

This is an alternate representation for Amdahl's Law. It is clearly the same thing as before just using S as the fraction of the program that is spent in the sequential part of the program.

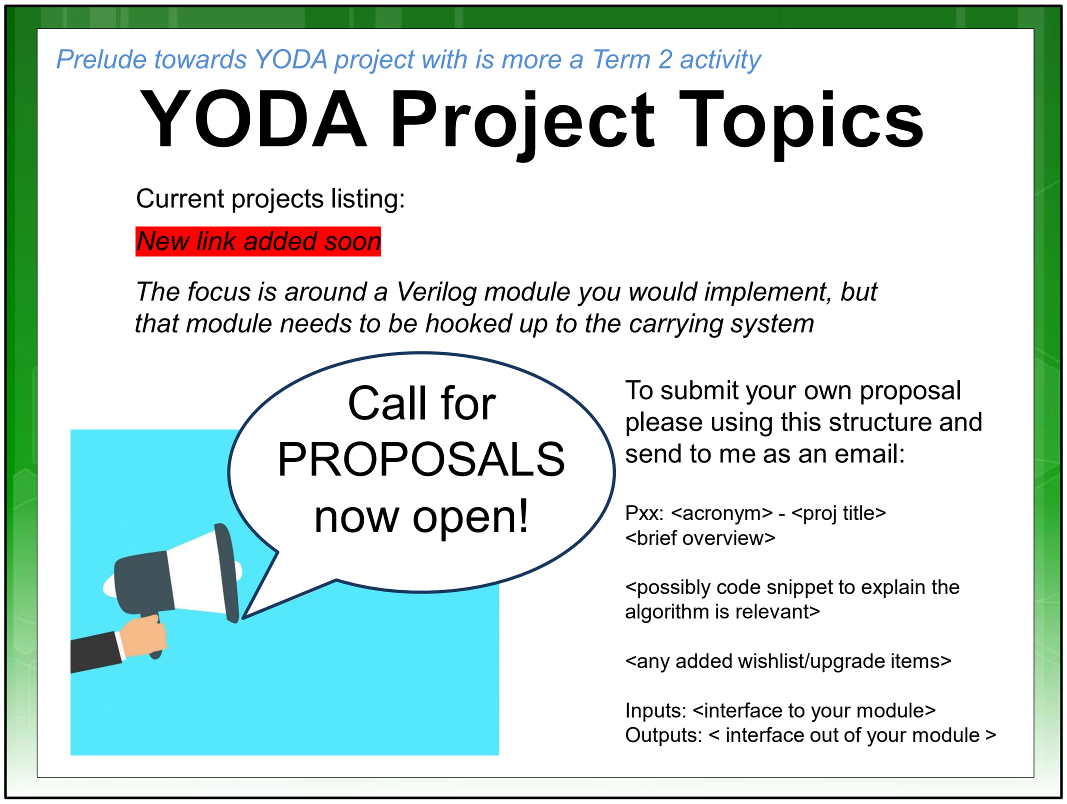
Homework task

Watch 2nd part of Amdahl's law video

[Understanding Parallel Computing (Part 2): The Lawn Mower Law LinuxMagazine]()

Amdahl2.flv

I do recommend watching this one as well, it is taking Amdahl's a bit further, perhaps simpler view (for gardeners especially), and suggests some drawbacks to the law also.



Prelude towards YODA project with is more a Term 2 activity

YODA Project Topics

Current projects listing:

New link added soon

The focus is around a Verilog module you would implement, but that module needs to be hooked up to the carrying system

To submit your own proposal please using this structure and send to me as an email:

Pxx: <acronym> - <proj title>
<brief overview>

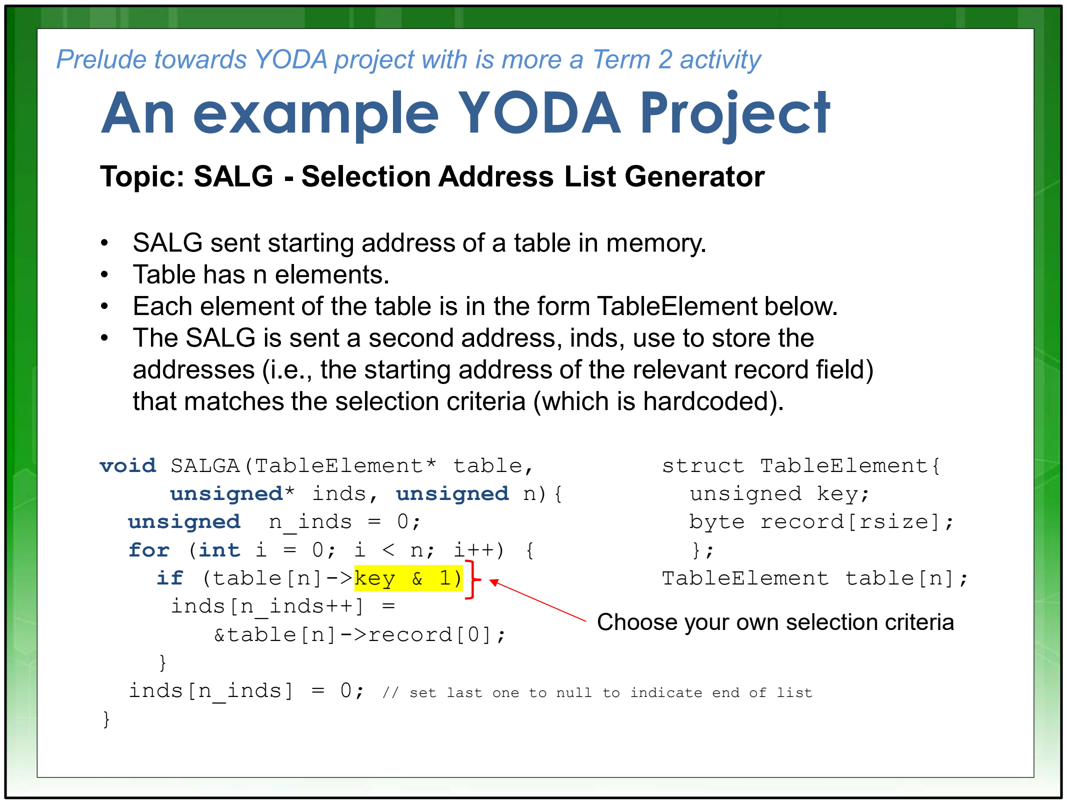
<possibly code snippet to explain the algorithm is relevant>

<any added wishlist/upgrade items>

Inputs: <interface to your module>
Outputs: < interface out of your module >

With all the talk earlier about product validation and teamwork, I couldn't resist putting in a suggestion here to anyone wanting to proposal their own YODA topic for next term (and perhaps save me time writing up topic ideas!).

Note that these are meant to be small experiments to explore performance boosting of a task using a parallel solution or specialized combinational logic hardware. Nothing too ambitious. Take a look at the current YODA project listing on the HPES website for ideas and examples of how the project briefs are given.



Prelude towards YODA project with is more a Term 2 activity

An example YODA Project

Topic: SALG - Selection Address List Generator

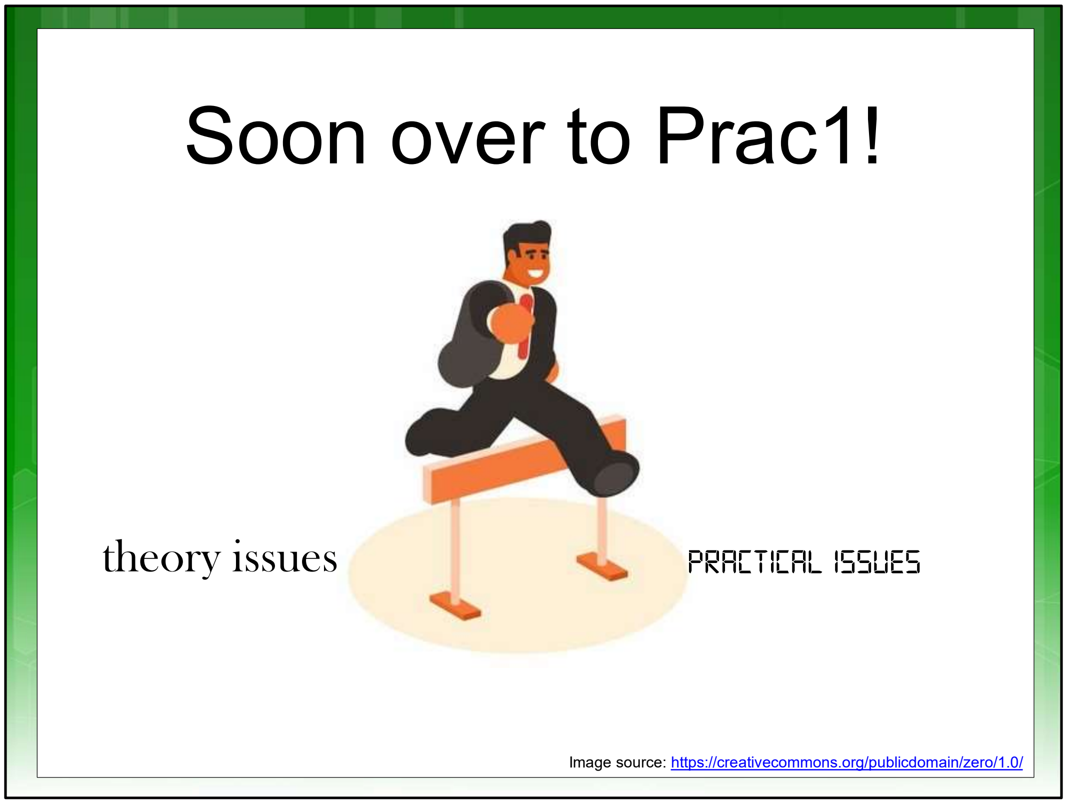
- SALG sent starting address of a table in memory.
- Table has n elements.
- Each element of the table is in the form TableElement below.
- The SALG is sent a second address, inds, use to store the addresses (i.e., the starting address of the relevant record field) that matches the selection criteria (which is hardcoded).

```
void SALGA(TableElement* table,
     unsigned* inds, unsigned n){
  unsigned  n_inds = 0;
  for (int i = 0; i < n; i++) {
    if (table[n]->key & 1)
     inds[n_inds++] =
        &table[n]->record[0];
    }
  inds[n_inds] = 0; // set last one to null to indicate end of list
}
```

```
struct TableElement{
  unsigned key;
  byte record[rsize];
  };
TableElement table[n];
```

Choose your own selection criteria

Here is an example of a YODA project topic. You and or your prospective YOG (i.e. YODA Group) need to prepare something like this if you want to propose a topic.



Soon over to Prac1!

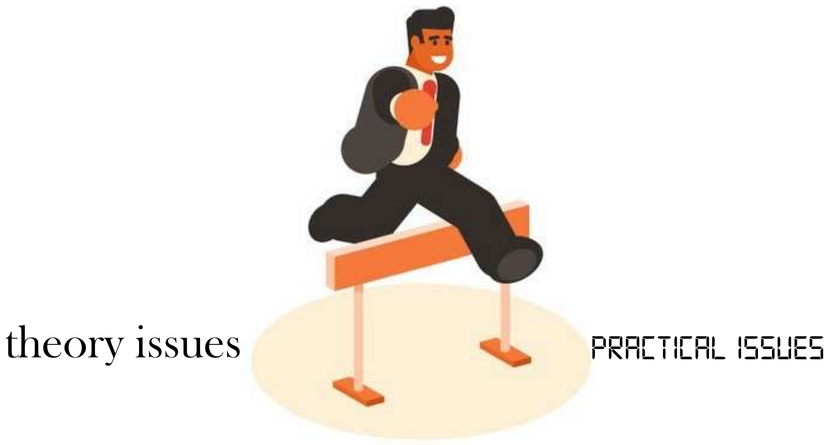


Image source: https://creativecommons.org/publicdomain/zero/1.0/

Time to get ready to transition over to the prac.. but will end with reminder about reading

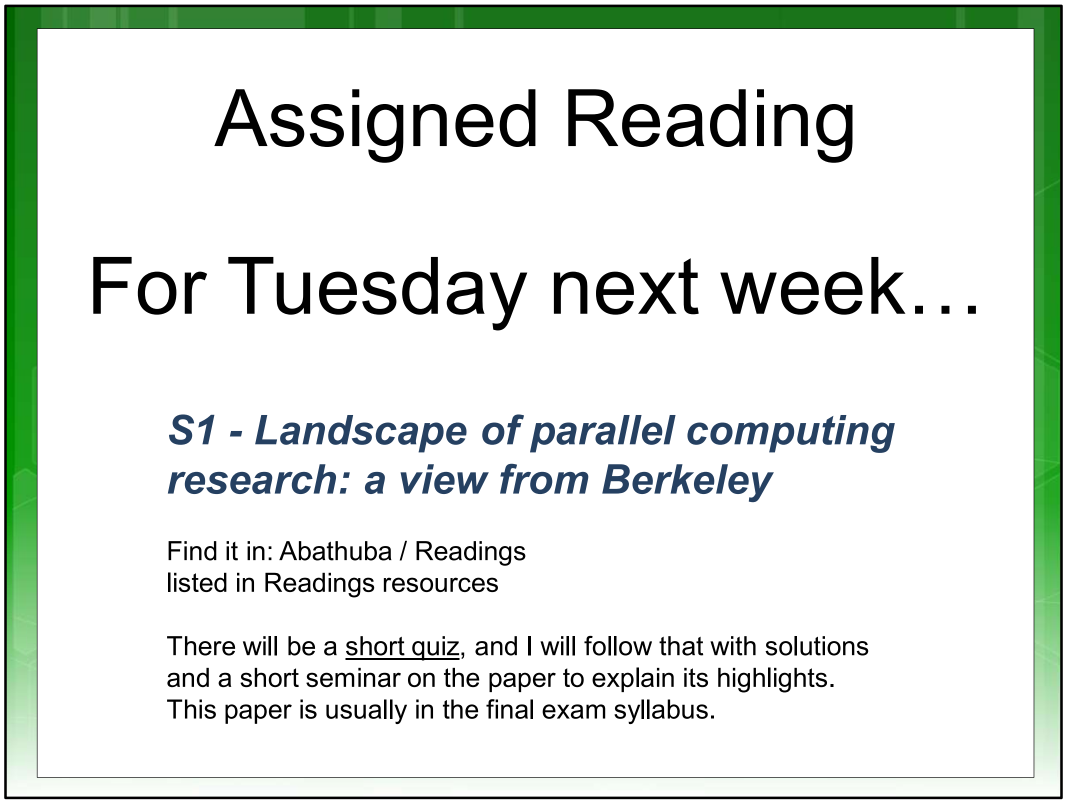
closing remarks & reminders…

Dealing with reading assignments

- You are suppose to read (at least speed read) the readings assigned as recommended – the others are more for deepening your insights into an area

```
open("L01 Berkeley 2006 - Landscale of
Parallel Computing Research.pdf") do pg1-8
```

… you might need to do more readings than that



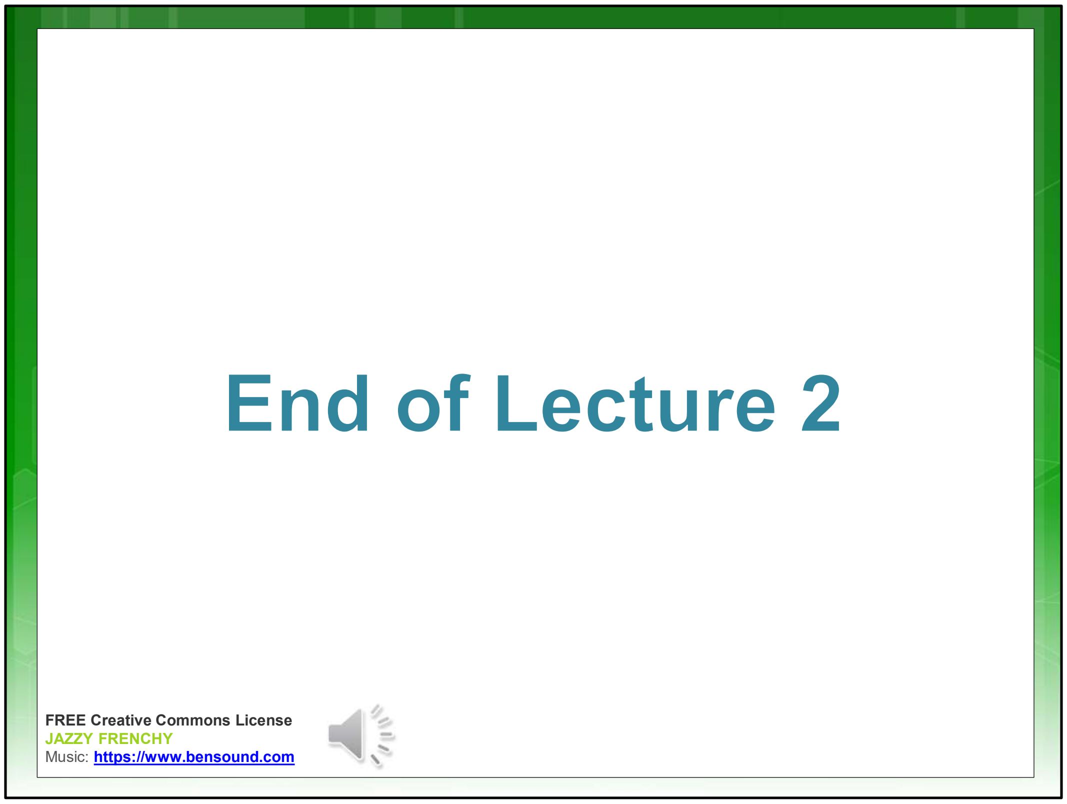
Assigned Reading

For Tuesday next week…

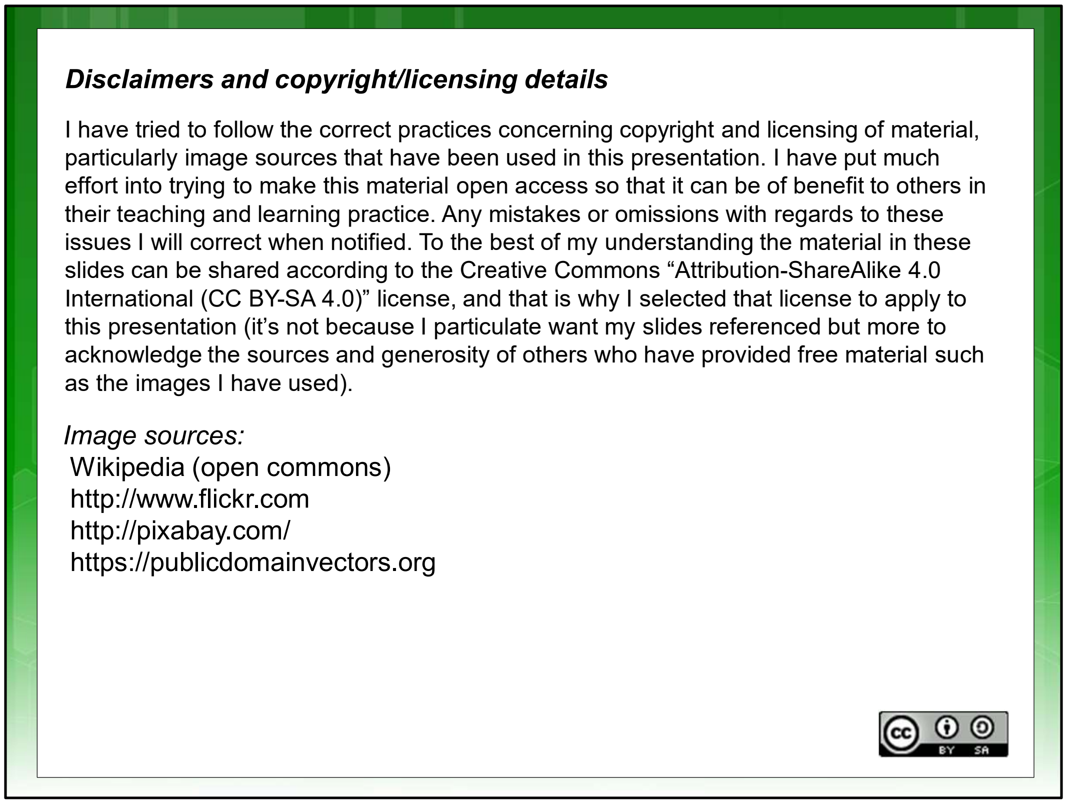
S1 - Landscape of parallel computing research: a view from Berkeley

Find it in: Abathuba / Readings
listed in Readings resources

There will be a short quiz, and I will follow that with solutions and a short seminar on the paper to explain its highlights.
This paper is usually in the final exam syllabus.



End of Lecture 2



Disclaimers and copyright/licensing details

Image sources:
Wikipedia (open commons)
http://www.flickr.com
http://pixabay.com/
https://publicdomainvectors.org