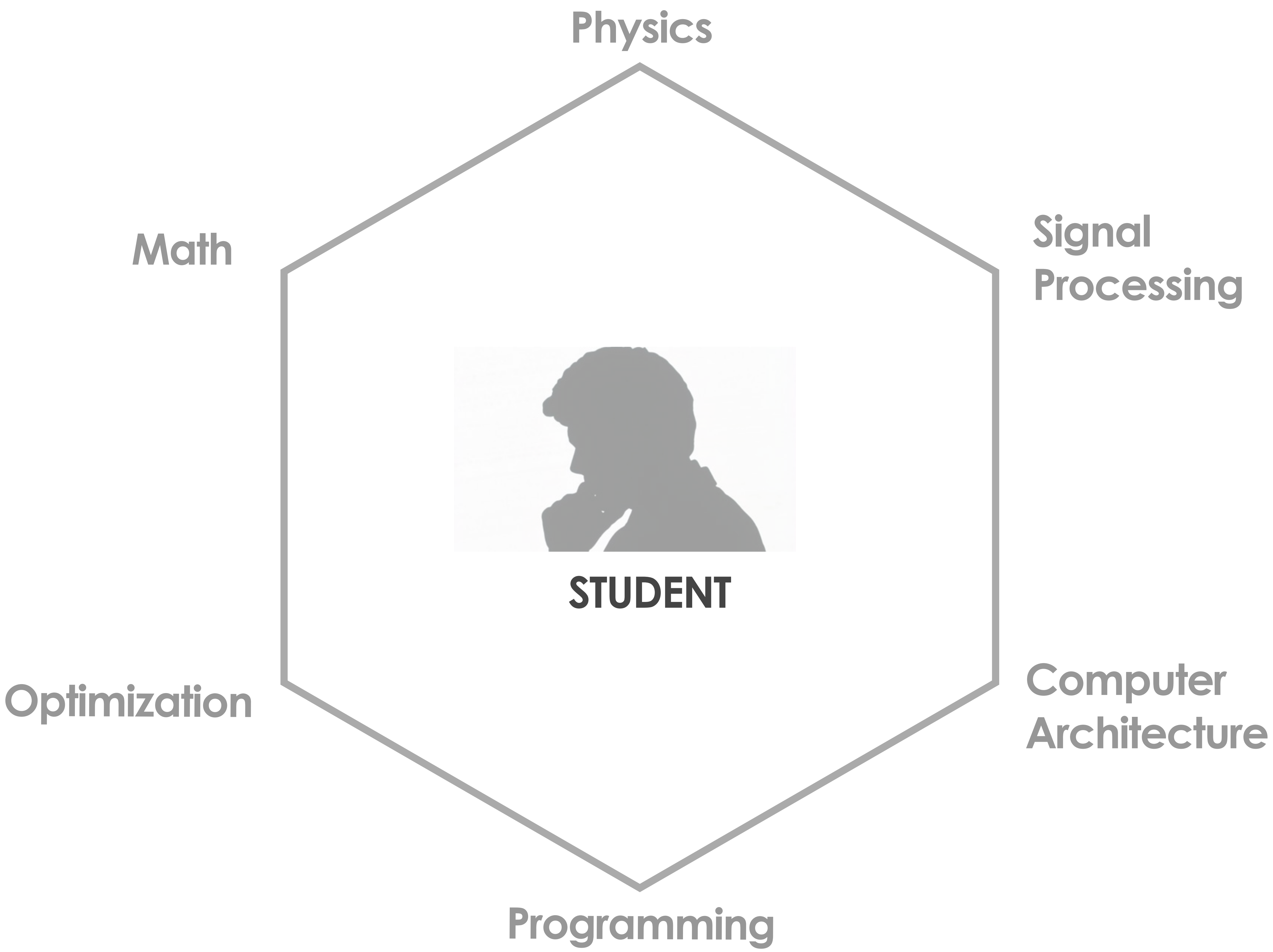
The **student-driven HPC** environment at **SLIM**

Tim Lin, Brazil IIP FWI Workshop 2015

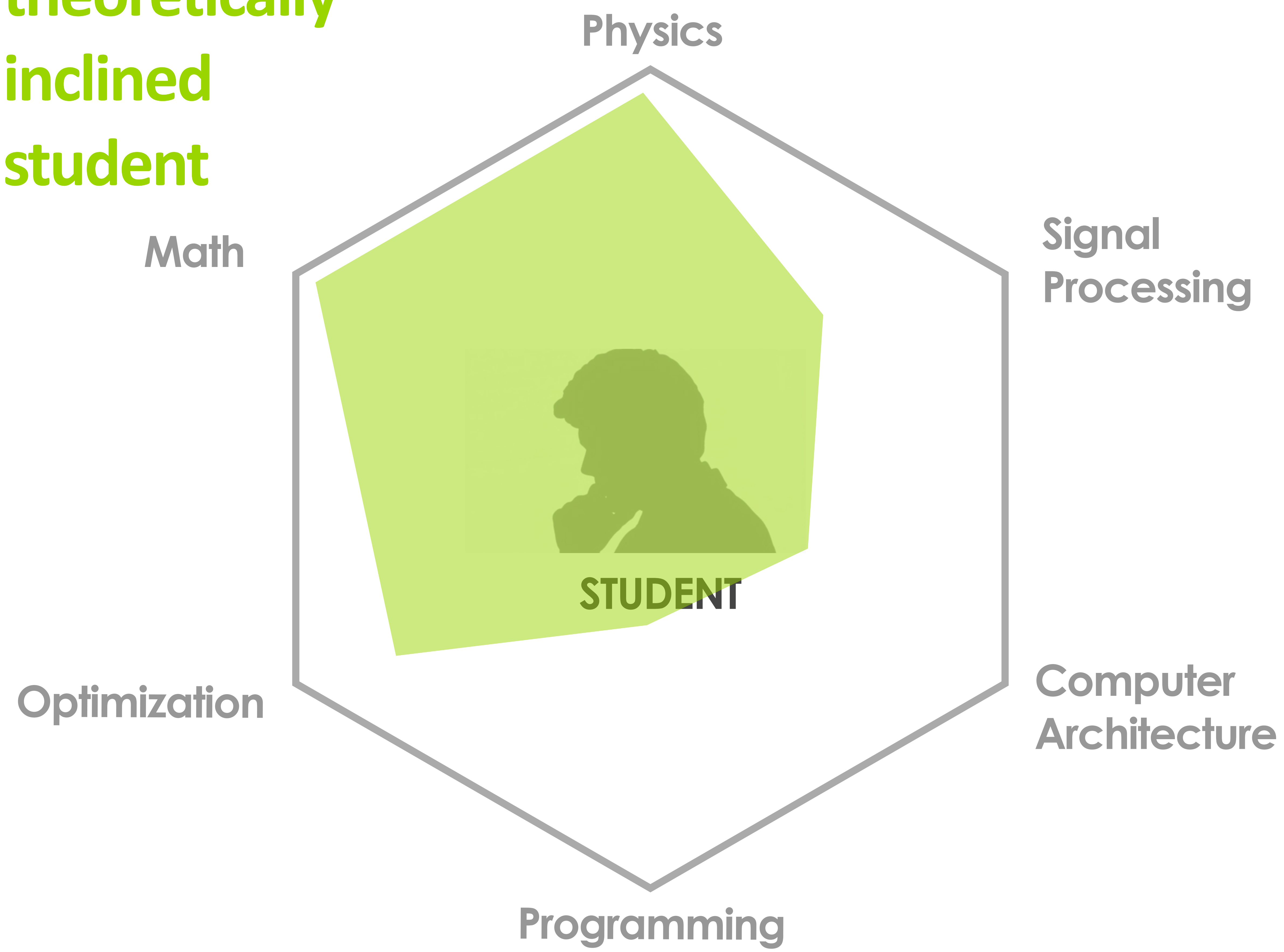
SLIM
University of British Columbia

Conservation of complexity...

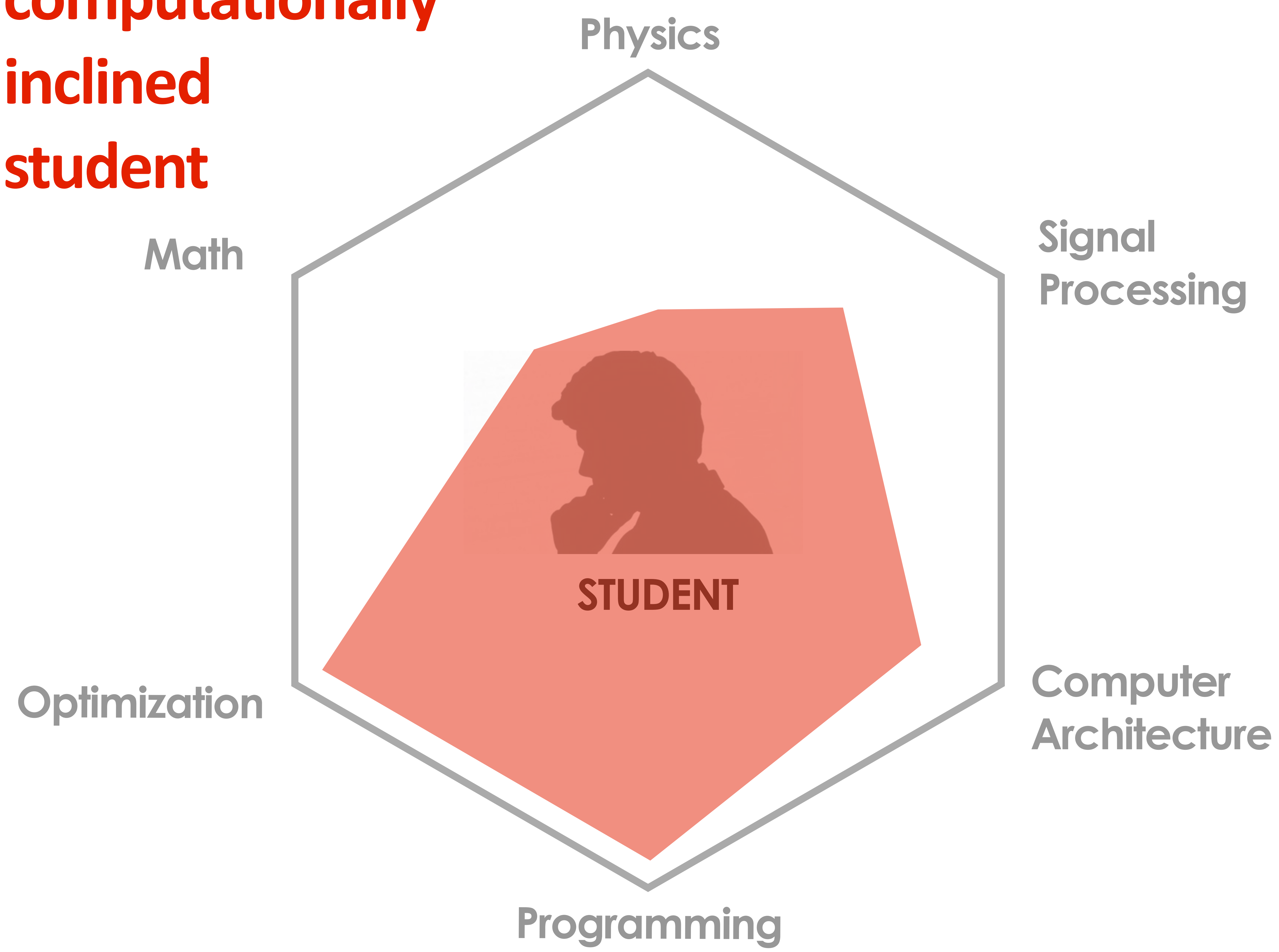
... in terms of a student's focus, attention, time

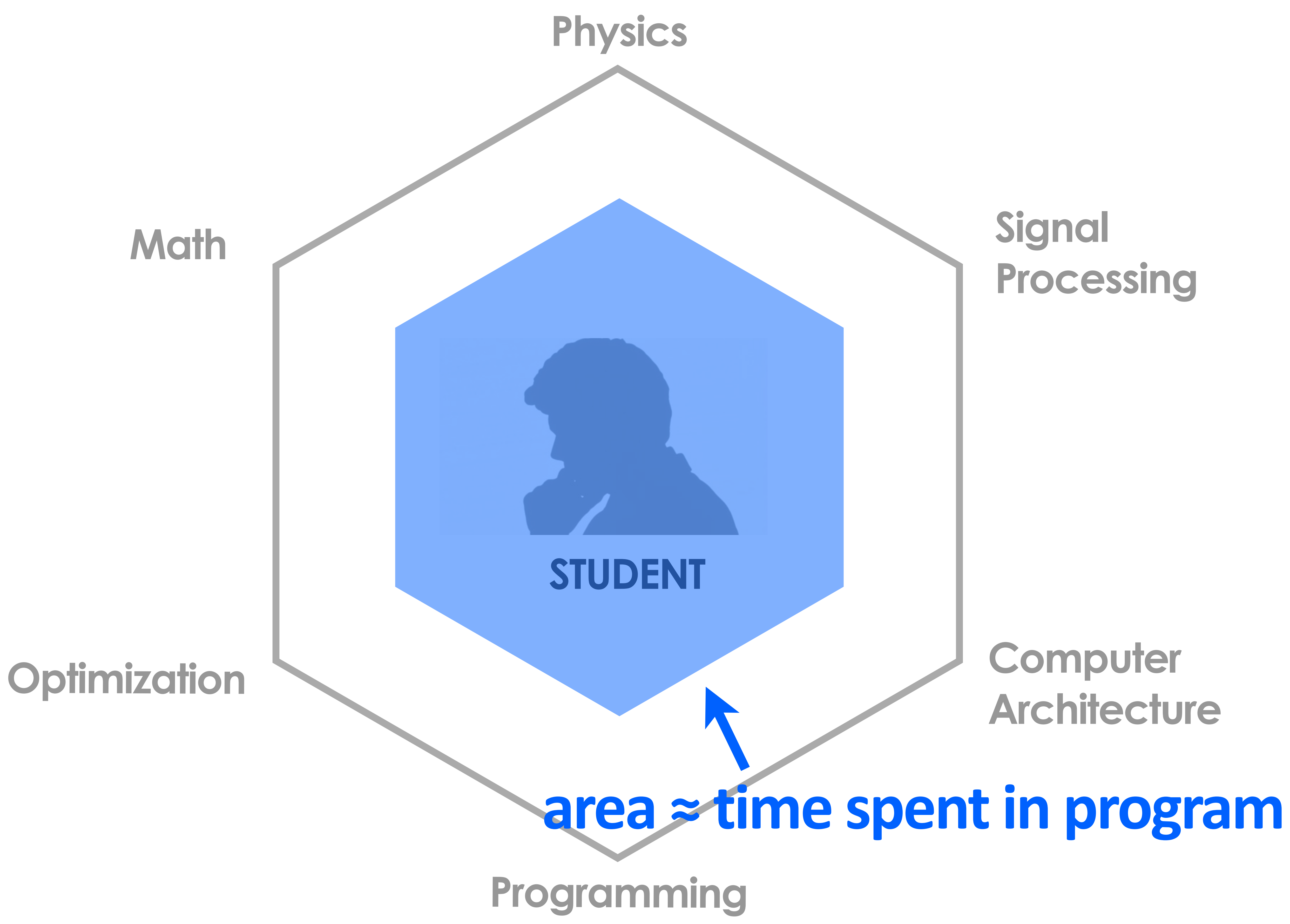

Physics
Math
Signal Processing
STUDENT
Optimization
Computer Architecture
Programming

theoretically
inclined
student

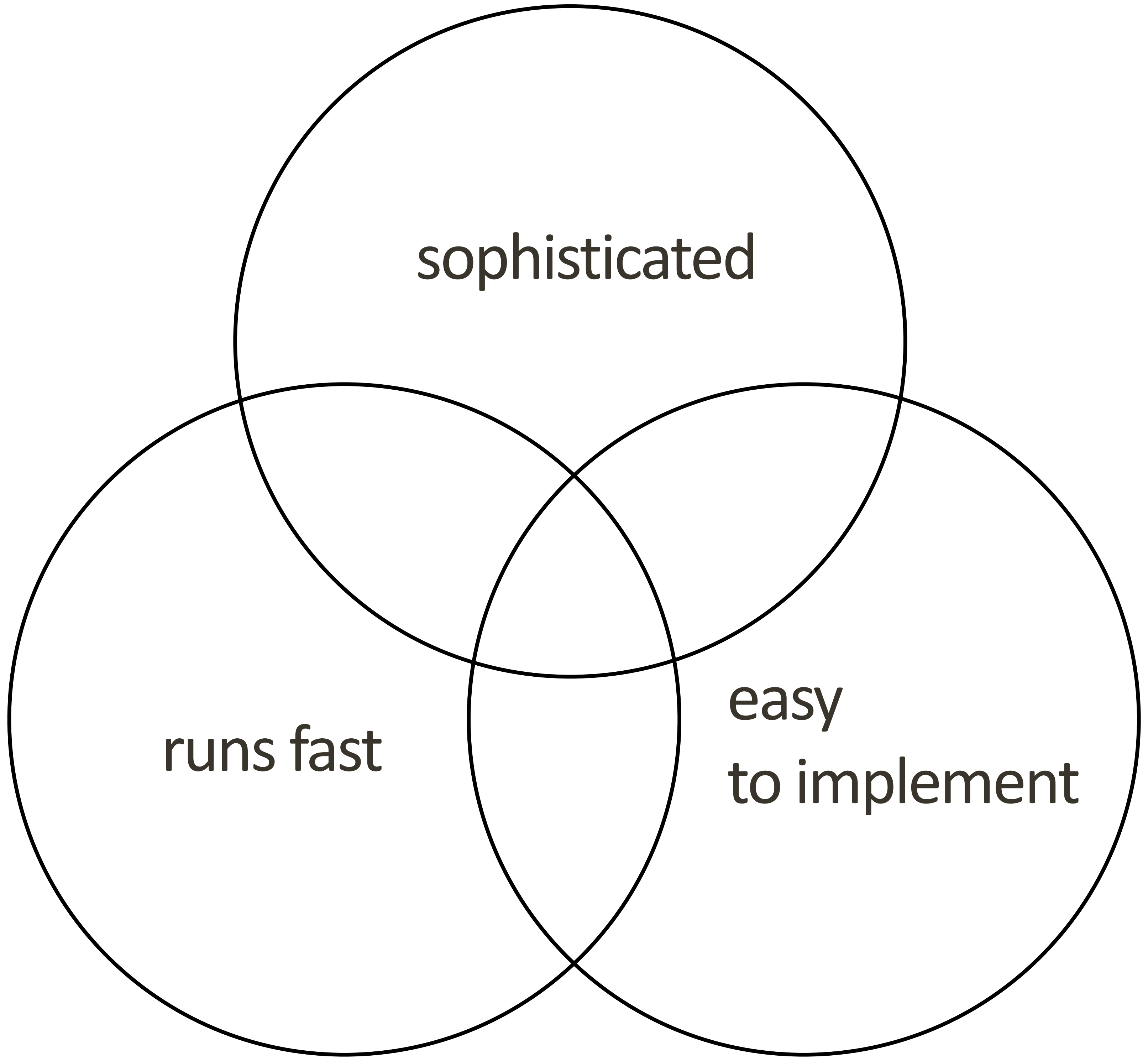


computationally
inclined
student



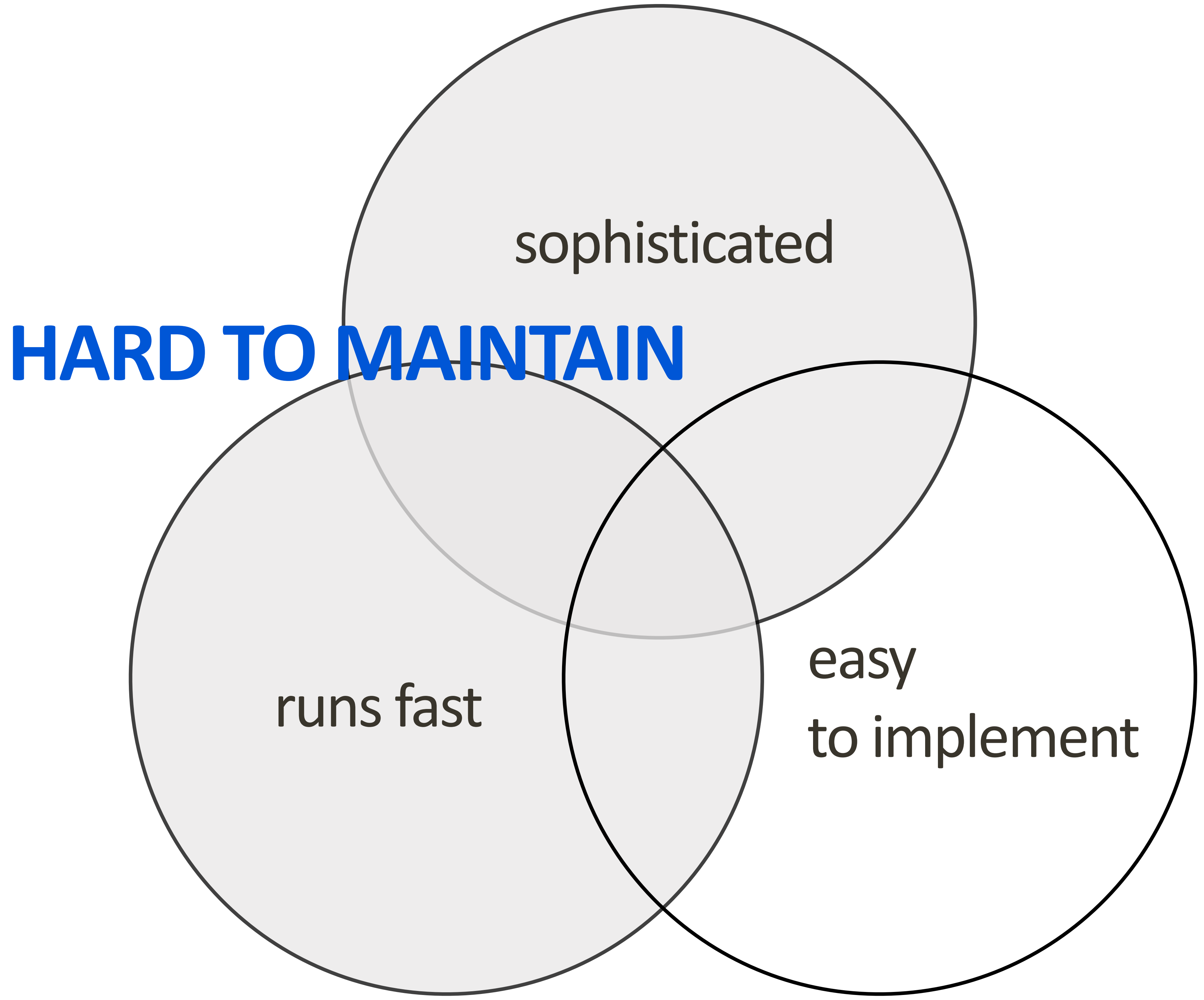

Physics
Math
Signal Processing
STUDENT
Optimization
Computer Architecture
area ≈ time spent in program
Programming

Trilemma of techniques explored



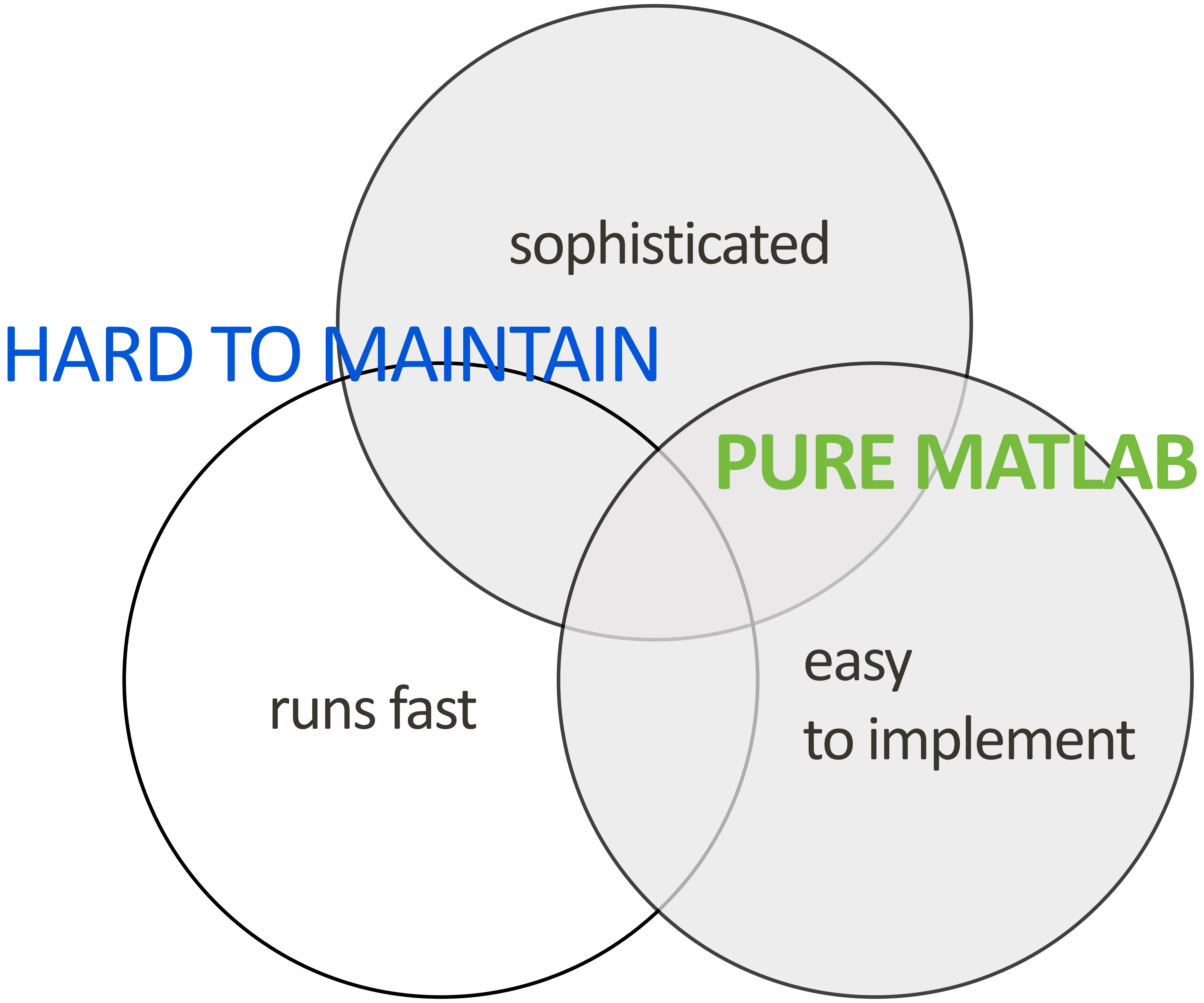
Choose 2

Trilemma of techniques explored



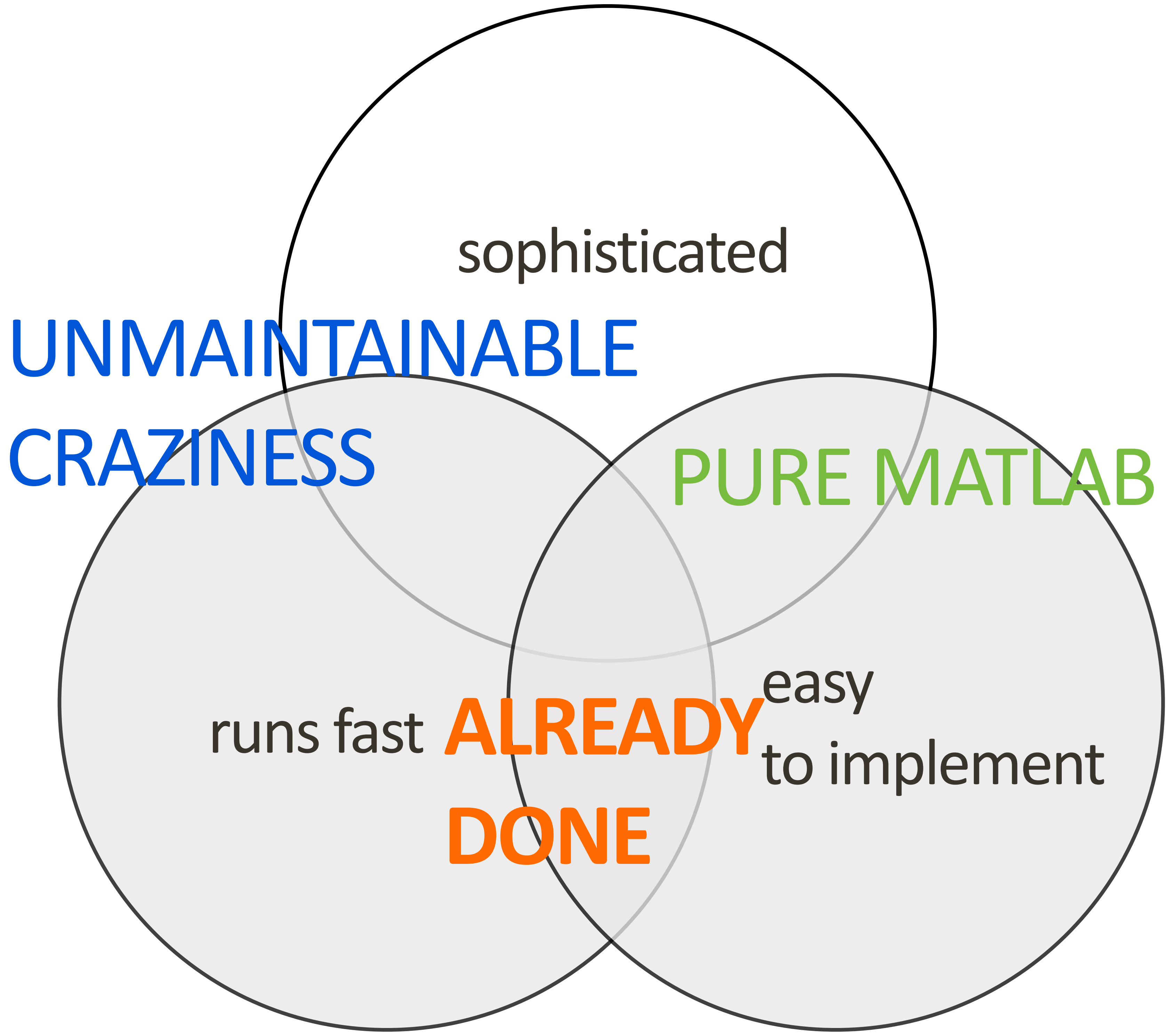
Choose 2

Trilemma of techniques explored



Choose 2

Trilemma of techniques explored



Choose 2

Pairing a “theoretical” and a “technical” person

Low-level fork

Projects get taken over and eventually stagnate

tweaks

different tweaks

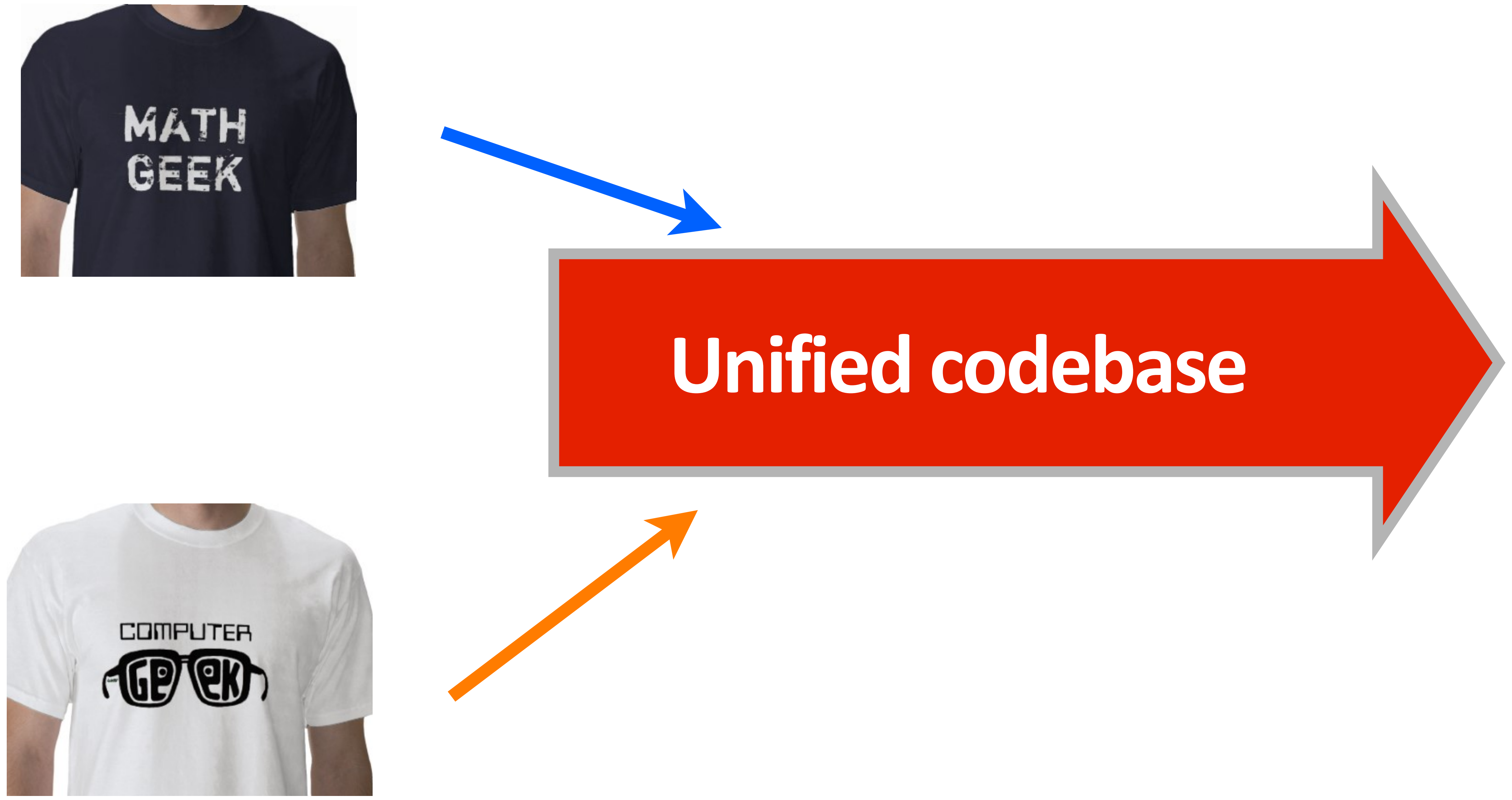
Projects get taken over and eventually stagnate

feature stagnation

Projects get taken over and eventually stagnate

low-level fork
unmaintainable

The “dream”



But which language/package?

Why use the same language or packages?

algorithm design

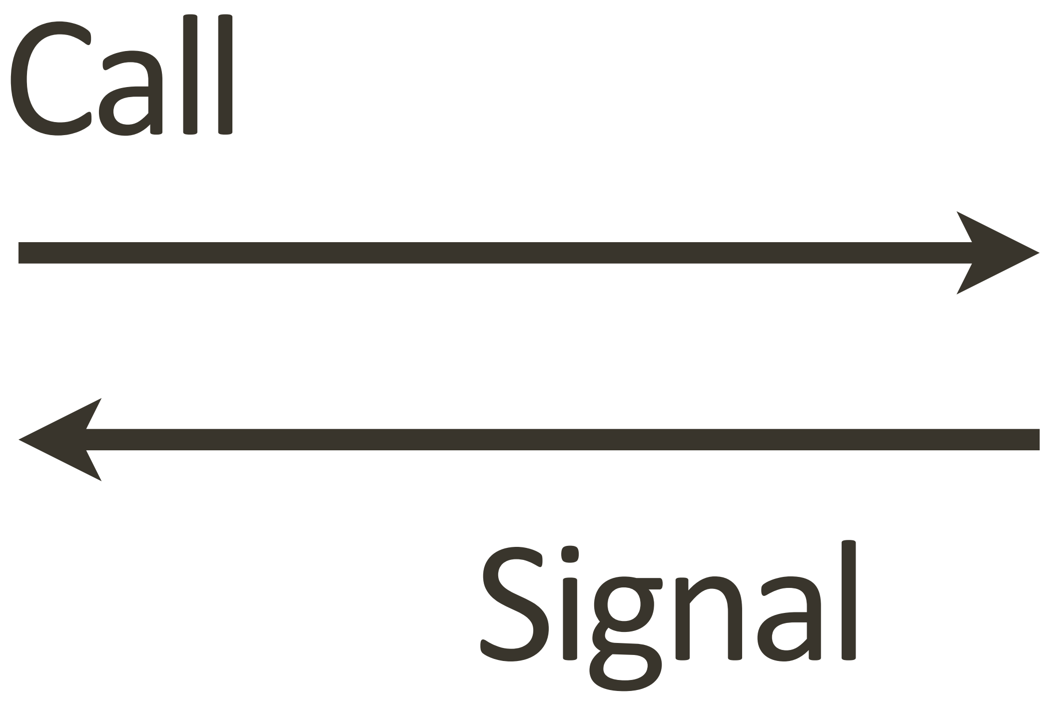
flexible
easy to read
easy to debug
reflects math
encourage experiments

computation engine

well-defined behaviour
mature compiler
low-level access
allows tweaking
parallel systems

algorithm design

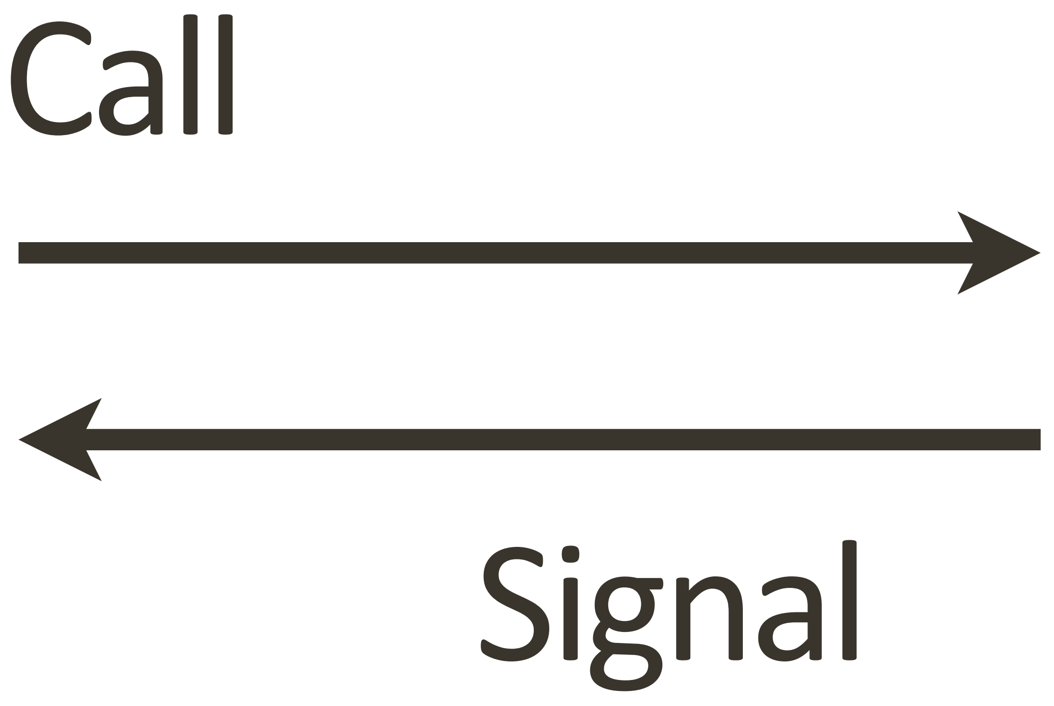
computation engine



algorithm
design

computation
engine

shell scripts



seismic utilities

algorithm
design

computation
engine

workflow
management

Call

Signal

computation
block

algorithm design

computation engine

C/C++/F90 → Call → BLAS/FFTW/MKL

C/C++/F90 ← Signal ← BLAS/FFTW/MKL

algorithm design

computation engine

MATLAB

Call →

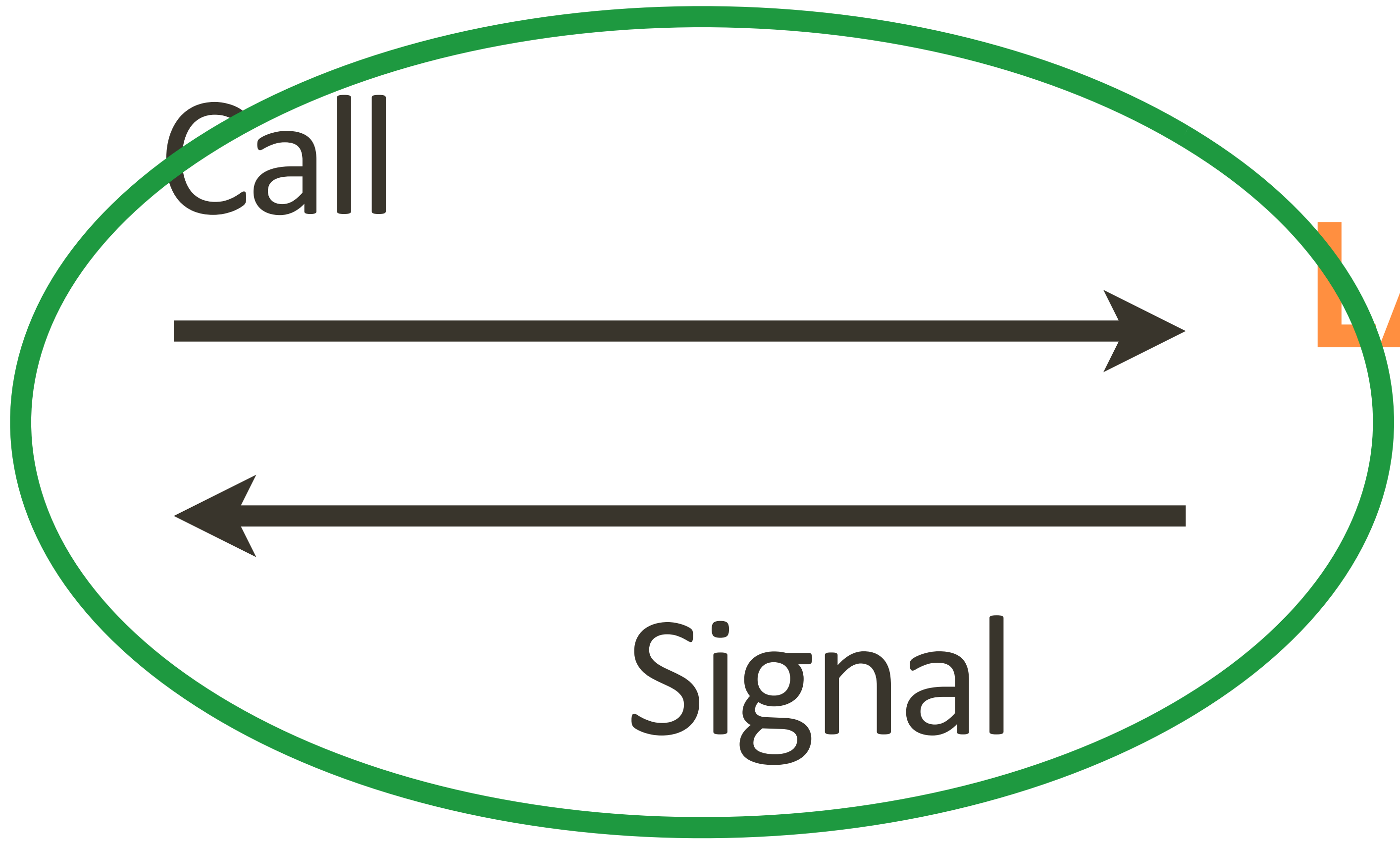
← Signal

LAPACK/ScaLAPACK/ MEX_files

algorithm
design

computation
engine

MATLAB



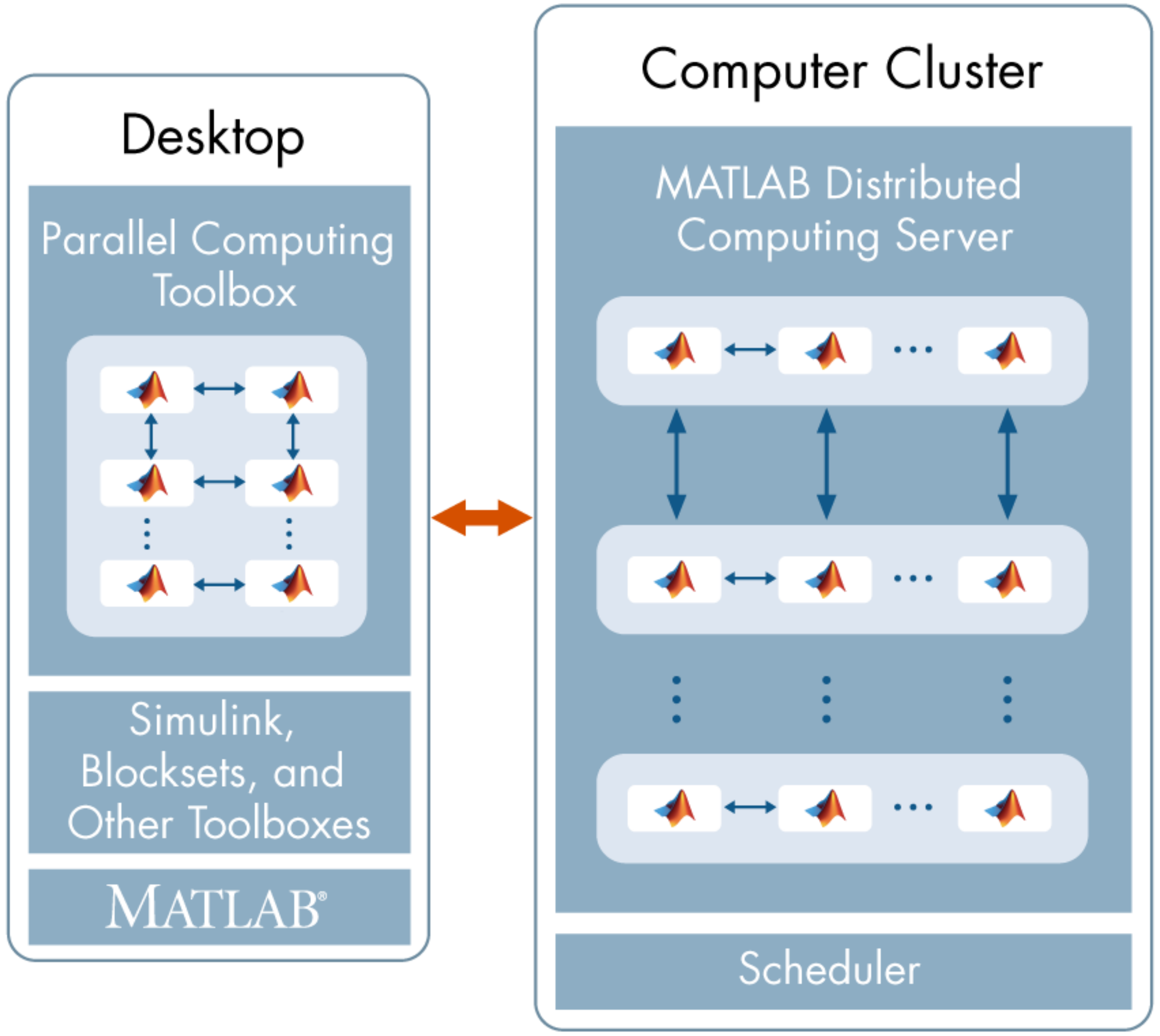
LAPACK/ScaLAPACK/
MEX_files

Abstraction layer is important

currently, we program in MATLAB, but our abstraction for distributed computation is based on

Parallel Matlab (PCT) **pSPOT**

"Parallel Matlab"



Officially another "toolbox" on top of Matlab, called "Parallel Computing Toolbox" or PCT

Two components:

- The "toolbox" itself, which provides the parallelization code and can spawn local workers
- The "Distributed Compute Server" (MDCS) which allows spawning workers on external nodes in a cluster
  - can bring own scheduler, i.e., SLIM uses Torque, SENAI uses Slurm

SLIM

The search for parallel arrays

Previous to 2009, SLIM was mostly based on SciPy/NumPy computing kernel with a traditional command-line seismic processing interface (at the time, RSF/Madagascar)

Developed a symbolic math DSL (SLIMpy) which translates mathematical expressions to an AST that writes out shell scripts that call RSF programs

- also write out reproducible SCons scripts (Make-equivalent)

But.... hard to parallelize non-trivially

The search for parallel arrays

Wanted: A true framework for shared-memory-like distributed arrays, which works similarly to NumPy arrays and is interactive

Two candidates

- Star-P (for Python)
- Matlab PCT (at the time just added distributed arrays)

Went with Matlab PCT on a hunch (actually it was cheaper)

- eventually Star-P acquired by Microsoft and “sunsetted”

Matlab PCT operation

Always assumes a “master” supervisor for a pool of workers

Each worker (and master) are independent, complete Matlab processes, and communicate via a MPI-based backend

Workers form a “pool” that can be provisioned and released interactively from the command line

Local workers free, individual licensing price for remote workers

Distributed arrays

Emulates a normal numeric array

- by default distributed evenly across the last dimension
- APIs to change underlying distribution
- can be constructed in many ways... from simple to complex
- easy way to learn about shared-memory/**NUMA** type architecture

Killer Feature: overloading of many Matlab functions on local numeric arrays to distributed arrays

abs
acos
acosd
acosh
acot
acotd
acoth
acsc
acscd
acsch
all
and(&)
angle
any
arrayfun
asec
asecd
asech
asin
asind
asinh
atan
cart2sph
cast
cat
ceil
cell2mat
cell2struct
celldisp
cellfun
char
chol
compan
complex
conj
corrcoef
cos
cosd
cosh
cot
cotd
coth
cov
csc
erfcx
erfinv
exp
expm1
eye
false
fieldnames
fft
fft2
fftn
find
fix
floor
full
gamma
gammainc
gammaincinv
gammaln
ge(>=)
gt(>)
hankel
horzcat([])
isinf
isinteger
islogical
isnan
isnumeric
isreal
issparse
ldivide(.\)
le(<=)
length
log
log10
log1p
log2
logical
lt(<)
lu
max
mean
median
meshgrid
min
numel
nzmax
ones
or(|)
permute
planerot
plus(+)
pol2cart
polyarea
polyval
pow2
power(.^)
prod
psi
qr
rand
randi
randn
rdivide(./)
real
reallog
realpow
sparse
spfun
sph2cart
spones
sqrt
std
struct2cell
subsasgn
subsindex
subsref
sum
svd
swapbytes
tan
tand
tanh
times(.*)
toeplitz
transpose(.')
trapz
tril
triu

atan
atan2
atan2d
atand
atanh
besselh
besseli
besselj
besselk
bessely
beta
betainc
betaincinv
betaln
bitand
bitor
bitxor
bsxfun
cart2pol

csc
cscd
csch
ctranspose(')
cummax
cummin
cumprod
cumsum
diag
diff
dot
double
eig
end
eps
eq(==)
erf
erfc
erfcinv

horzcat([])
hsv2rgb
hypot
ifft
ifft2
ifftn
imag
Inf
int16
int32
int64
int8
inv
ipermute
isempty
isequal
isequaln
isfinite
isfloat

min
minus(-)
mldivide(\)
mrdivide(/)
mtimes(*)
mod
mode
NaN
ndims
ndgrid
ne(~=)
nextpow2
nnz
nonzeros
norm
normest
not(~)
nthroot
num2cell

realpow
realsqrt
rem
repmat
reshape
rgb2hsv
rmfield
round
sec
secd
sech
sign
sin
sind
single
sinh
size
sort
sortrows

triu
true
typecast
uint16
uint32
uint64
uint8
uminus(-)
unwrap
uplus(+)
vander
var
vertcat([;])
xor
zeros

Distributed arrays

Allows many existing codebase to work directly on distributed arrays with very few changes

- improved collaboration with outsiders
- brute effort provided by Mathworks, continuous improvement
- trading licensing fee for student time
- vastly improved maintainability from being able to limit code branching for parallel mode

“Not that slow”

HPCC High-Performance Linpack benchmark in three lines:

```
A = distributed.randn(m, m, distributor2dbc);
b = distributed.rand(m, 1);
tic
x = A\b;
toc;
```

Just ran small benchmark on YEMOJA:

- 128 nodes, 8 process (1024 total)
- inverting 1,800,000-by-1,800,000 matrix (380 GB)
- **took 623 seconds, about 12 TFlop/s**

“Not that slow”

Verification code as per HPCC spec

```
% Compute scaled residuals
r1 = norm(A*x-b,inf)/(eps*norm(A,1)*m);
r2 = norm(A*x-b,inf)/(eps*norm(A,1)*norm(x,1));
r3 = norm(A*x-b,inf)/(eps*norm(A,inf)*norm(x,inf)*m);

if max([r1 r2 r3]) > 16
    error('Failed the HPC HPL Benchmark');
end
```

SLIM

SPOT/pSPOT built on distributed array

A way to encapsulate kernel computations of linear operations into something that “looks like a matrix”

```
F = opDFT(512)
x = randn(512,1)
xf = F * x
x == F’ * xf
```

Inherently express the notion of multilinear transformations on tensors into Kronecker products

```
FK = opKron(opDFT(300), opDFT(512))
x = randn(512,300)
x_fk = FK * x(:)
x == F’ * xf
```

SPOT/pSPOT built on distributed array

Extends to distributed paradigm (implicitly performs transpose)

```
F = opDFT(1024);
F2D = opKron(F,F);
F4D = oppKron2Lo(F2D,F2D);
F5D = oppKron2Lo(F2D,opKron(F,F,F));

x = distributed.randn(1024*1024*1024,1024*1024);
xf = F5D * x(:);
```

Non-separable example

Frequency-dependent filtering

```
A = oppDistFun(f,@filter)
```

f	is (distributed) array of frequencies
@filter(x,f)	performs filter on x based on frequency f

Slice-wise matrix-matrix multiply

```
A = oppDistFun(MAT,@matmult)
```

MAT	is 3D array distributed over the “slice” dim
@matmult(x,mat)	performs mat-mult between x and mat

Non-separable example

Implemented SRME multiple prediction step using this framework

Just did small benchmark on YEMOJA again

- 128 nodes, 8 process (1024 total)
- Seismic line data: 2200 time samples, 2200 shots, 2200 trace/shot
- ~ 42GB of data in single precision
- SRME prediction finished in 751 seconds (~ 70 GFlop/s) using FFT

Parfor

A simple way to do parallel loops

Concept of a parallel index variable, Matlab AST parser will enforce that you do not use it to index

- *except* for associative reduction operations and functions
- a simple way to learn about map reduce for students
- latest Matlab can also connect to Hadoop for “real” mapreduce

SPMD

Single program, multiple data paradigm

- Each worker has local execution space using variable of same name
- Master has access to all worker's local results outside of SPMD context, workers can also communicate
- Very easy to establish barriers and broadcasts
- Easy way for students to work in **UPC/BSP** paradigm, with **MPI** primitive equivalents available
- works well with distributed arrays (can access local part)

available from the parallel pool. If there are not enough workers available, an error is thrown. If n is zero, MATLAB executes the bl
and creates Composite objects, the same as if there is no pool available.

spmd(m,n), statements, end uses a minimum of m and a maximum of n workers to evaluate statements. If there are not e
available, an error is thrown. m can be zero, which allows the block to run locally if no workers are available.

For more information about spmd and Composite objects, see Distributed Arrays and SPMD.

Examples

Perform a simple calculation in parallel, and plot the results:

```
parpool(3)
spmd
  % build magic squares in parallel
  q = magic(labindex + 2);
end
for ii=1:length(q)
  % plot each magic square
  figure, imagesc(q{ii});
end
delete(gcp)
```

More About

Tips

- An spmd block runs on the workers of the existing parallel pool. If no pool exists, spmd will start a new parallel pool, unless tl
starting of pools is disabled in your parallel preferences. If there is no parallel pool and spmd cannot start one, the code runs

Detailed communication control btw workers

Many message-passing communication routines from MPI are exposed in a Matlab way

Task Control and Worker Communication

Control task code execution and communication among workers during job and spmd block execution

Functions

`labindex`	Index of this worker
`numlabs`	Total number of workers operating in parallel on current job

`gcat`	Global concatenation
`gop`	Global operation across all workers
`gplus`	Global addition

`pload`	Load file into parallel session
`psave`	Save data from communicating job session

`labBarrier`	Block execution until all workers reach this call
`labBroadcast`	Send data to all workers or receive data sent to all workers
`labProbe`	Test to see if messages are ready to be received from other worker
`labReceive`	Receive data from another worker
`labSend`	Send data to another worker
`labSendReceive`	Simultaneously send data to and receive data from another worker

`getCurrentJob`	Job object whose task is currently being evaluated
`getCurrentCluster`	Cluster object that submitted current task
`getCurrentTask`	Task object currently being evaluated in this worker session

```
FUN(FUN(x1,x2),x3) = FUN(x1,FUN(x2,x3))
```

`res = gop(FUN,x,targetlab)` performs the reduction, and places the result into `res` only on the worker in

Examples

This example shows how to calculate the sum and maximum values for x among all workers.

```
p = parpool('local',4);
x = Composite();
x{1} = 3;
x{2} = 1;
x{3} = 4;
x{4} = 2;
spmd
    xsum = gop(@plus,x);
    xmax = gop(@max,x);
end
xsum{1}
```

```
10
```

```
xmax{1}
```

```
4
```

This example shows how to horizontally concatenate the column vectors of x from all workers into a matrix. It u

SLIM

Conclusions

- Student time remains constant over the years, but things to learn increase faster and faster each year
- Abstractions save us from inevitable specialization
- Using high-level abstraction and easy tools for parallel computation, students can save programming time and use it on other topics that are also becoming increasingly complex
- Encourages collaboration by minimizing mundane parts of the codebase, many cases serial and parallel program can share the same codes
- *Not* for free: enforces good programming style and separation of concern for the code
- This paradigm great increased scientific productivity at SLIM