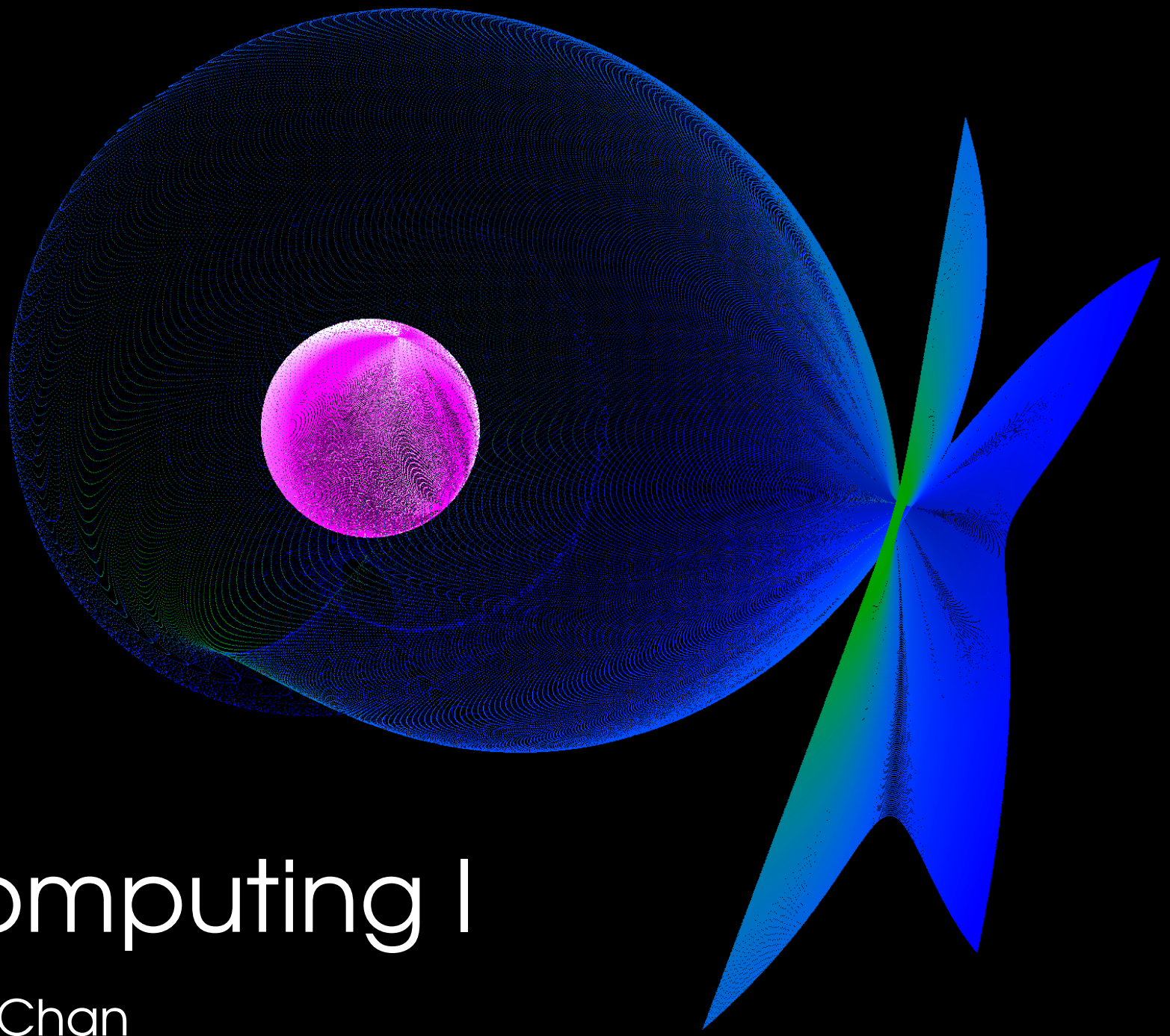
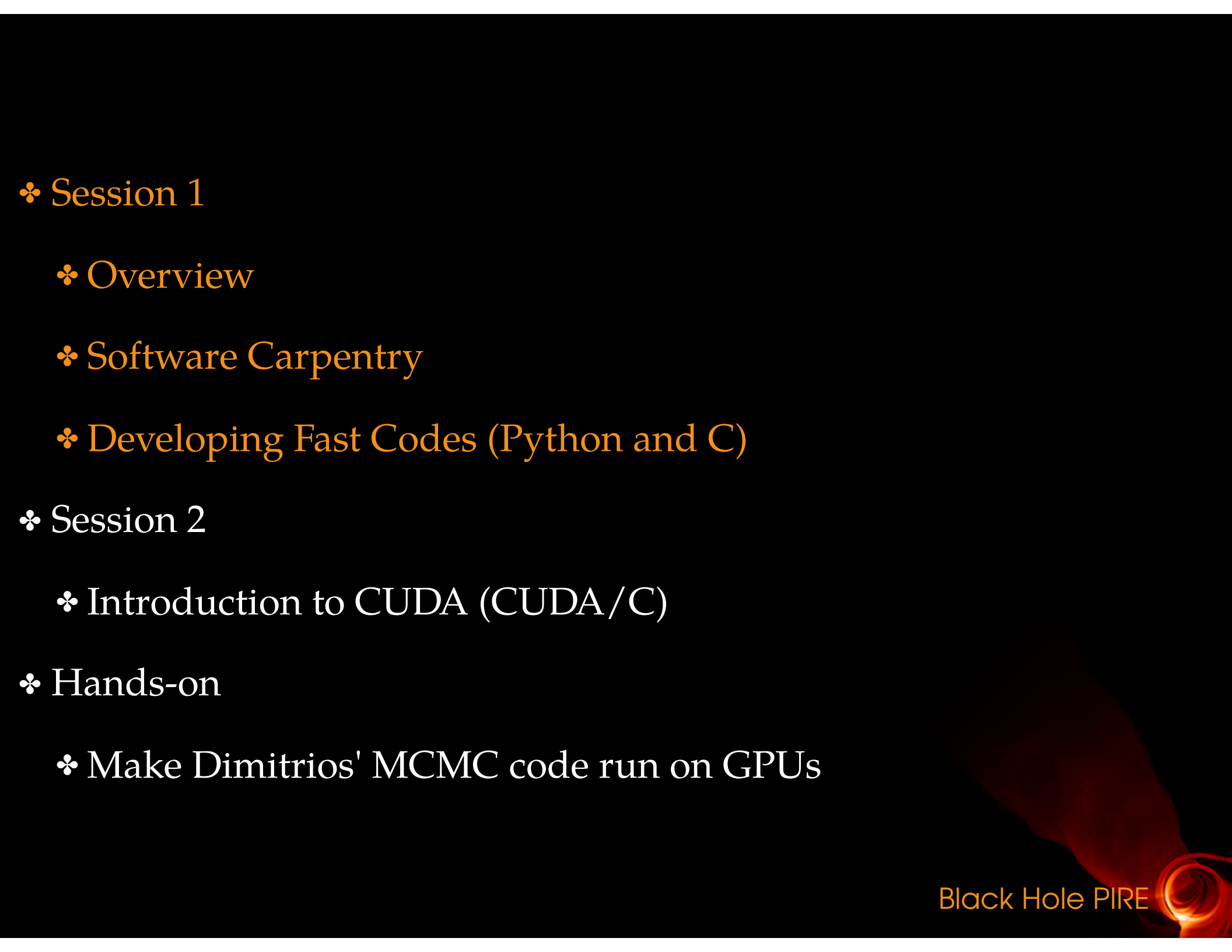


GPU Computing I

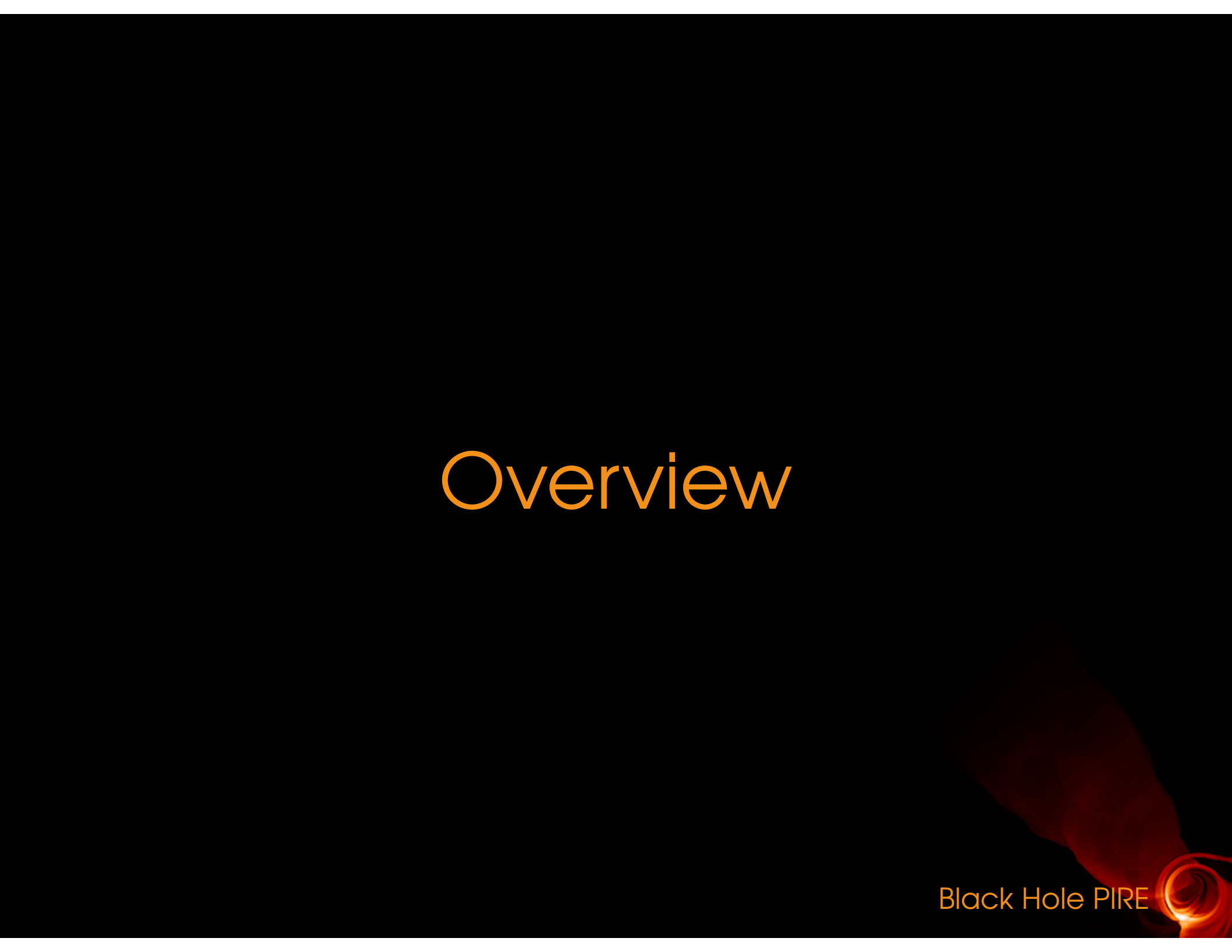
Chi-kwan "CK" Chan

Dec 5th, 2018, Biosphere2, PIRE Winter School

- Session 1
  - Overview
  - Software Carpentry
  - Developing Fast Codes (Python and C)
- Session 2
  - Introduction to CUDA (CUDA/C)
- Hands-on
  - Make Dimitrios' MCMC code run on GPUs



Overview



From Video Games to Simulations

From Video Games to Simulations

From Video Games to Simulations

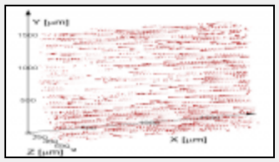
CUDA Community Showcase

https://www.nvidia.com/object/cuda_showcase_html.html

CUDA Community Showcase - HTML Version

GPU computing applications developed on the CUDA architecture by programmers, scientists, and researchers around the world.

Click here for the Flash version



Flow dynamics measurements using digital holographic PIV

An in-line digital holographic (D-HPIV) setup and CUDA-accelerated algorithm were implemented in order to measure the instantaneous three-dimensional (3D), three-component (3C) velocity field of nonstationary flows. This increases dramatically the speed of digital video hologram processing. The system can measure the number, 3D position, size, 3C velocity and track of the particles. The results of the hologram reconstruction are represented using OpenGL.

Speed Up: 1000 X

Paper

1000x

Author(s)	Dmitry Ekimov	**Organization Type**	Academia	**Organization**	Petrozavodsk State University	**Platfc**
Software License	N/A	**Application Type**	N/A	**Date Released**	10/07/2010	

Numerical simulation of flow around an oscillating cylinder

This program presents a finite difference solution for 2D, low Reynolds number (1-350), unsteady flow around and heat transfer from a stationary or oscillating circular cylinder with constant surface temperature and placed in a uniform stream. The fluid is assumed to be incompressible and of constant property. The cylinder is moved mechanically and can vibrate in-line with or transverse to the main stream or can follow an elliptical or figure-8-shaped path. The governing equations are the Navier-Stokes equations, the continuity equation, a Poisson equation for pressure and the energy equation.

Speed Up: 13 X

Multime

13x

Author(s)	Prof. Laszlo Baranyi, Laszlo Daroczy	**Organization Type**	Academia	**Organization**	Department of Fluid and Heat Engineering, University of Miskolc, Hungary	**Platfc**
Software License	N/A	**Application Type**	N/A	**Date Released**	02/02/2010	

From Video Games to Simulations

UA's El Gato Supercomputer

Tesla
CUDA

Fermi
FP64

Kepler
Dynamic Parallelism

Maxwell
DX12

Pascal
Unified Memory

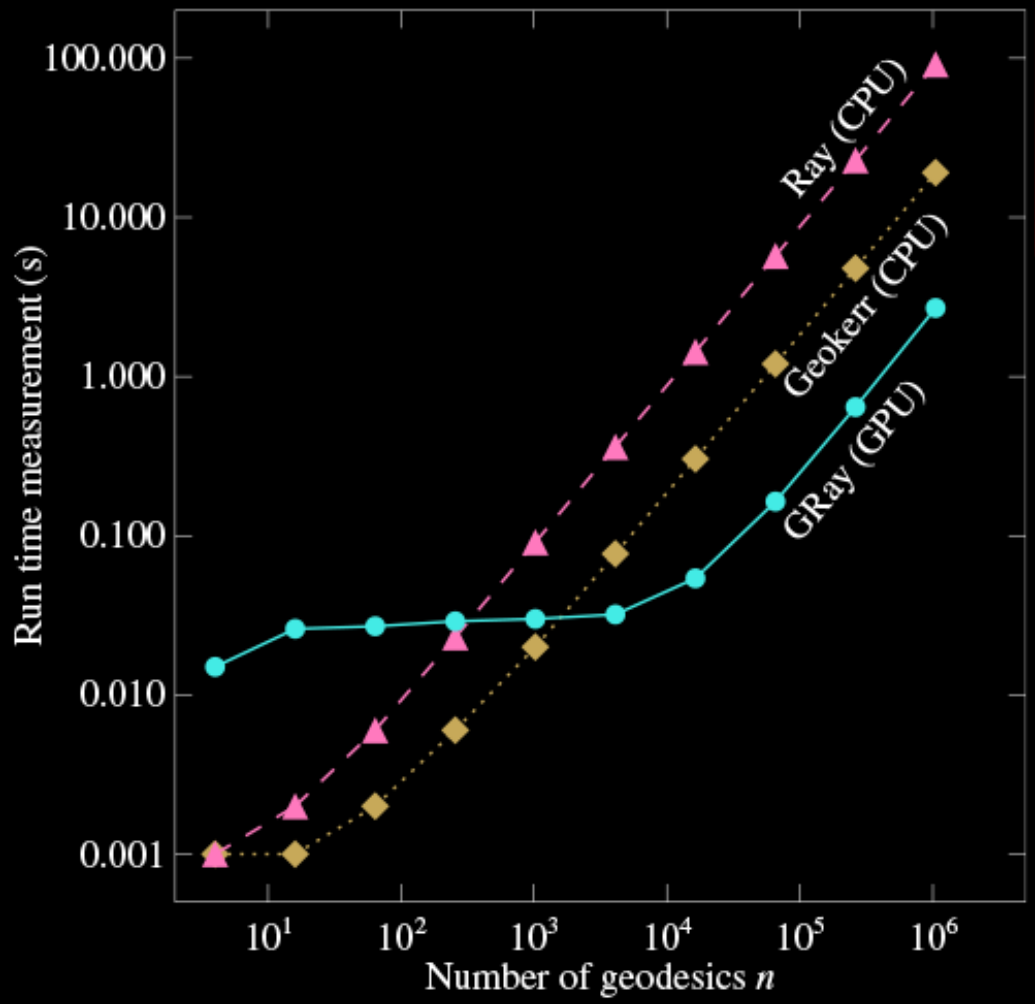
UA's Ocelote Supercomputer

2008 2010 2012 2014 2016

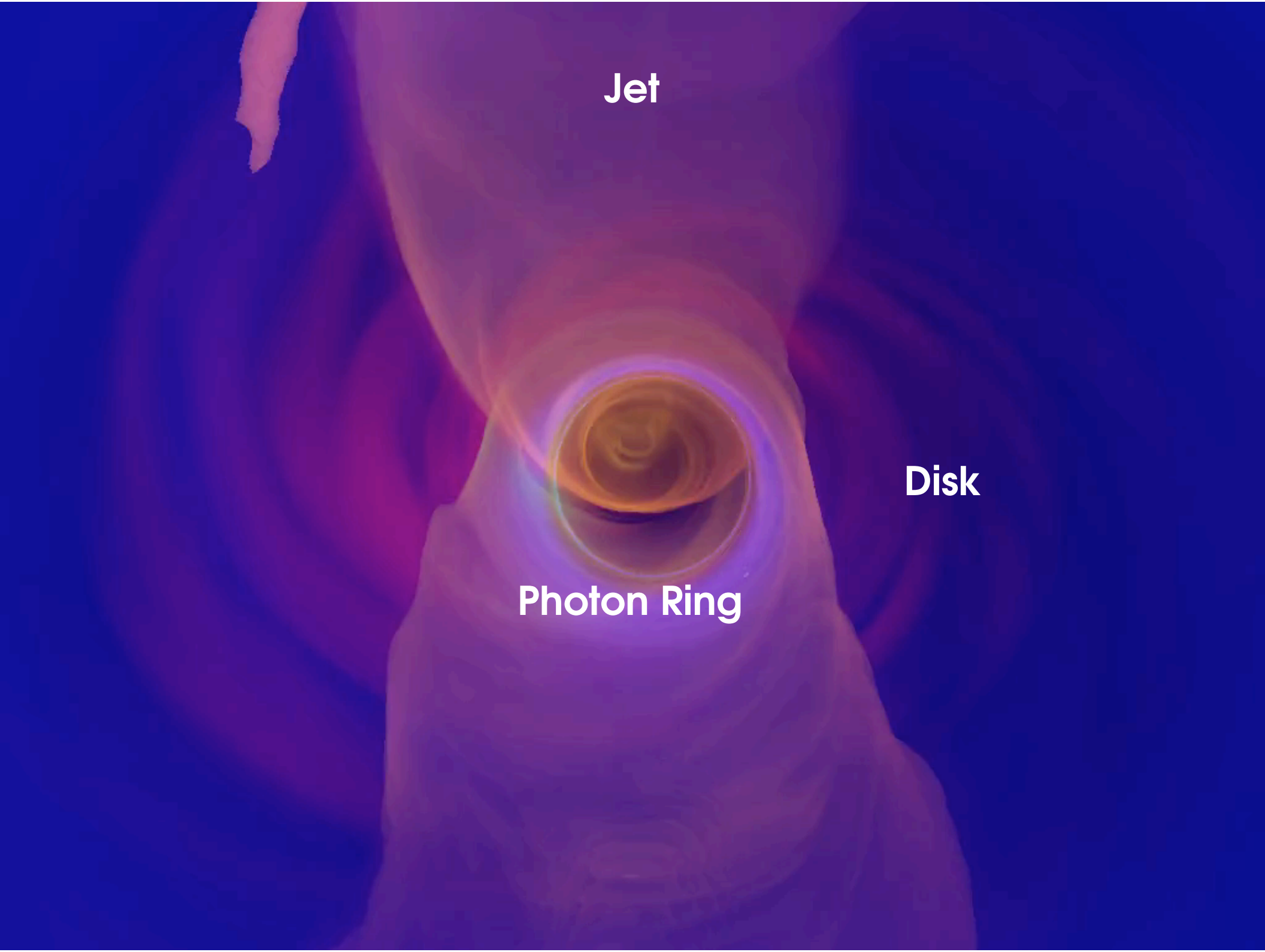
Portable & Adaptive Performance

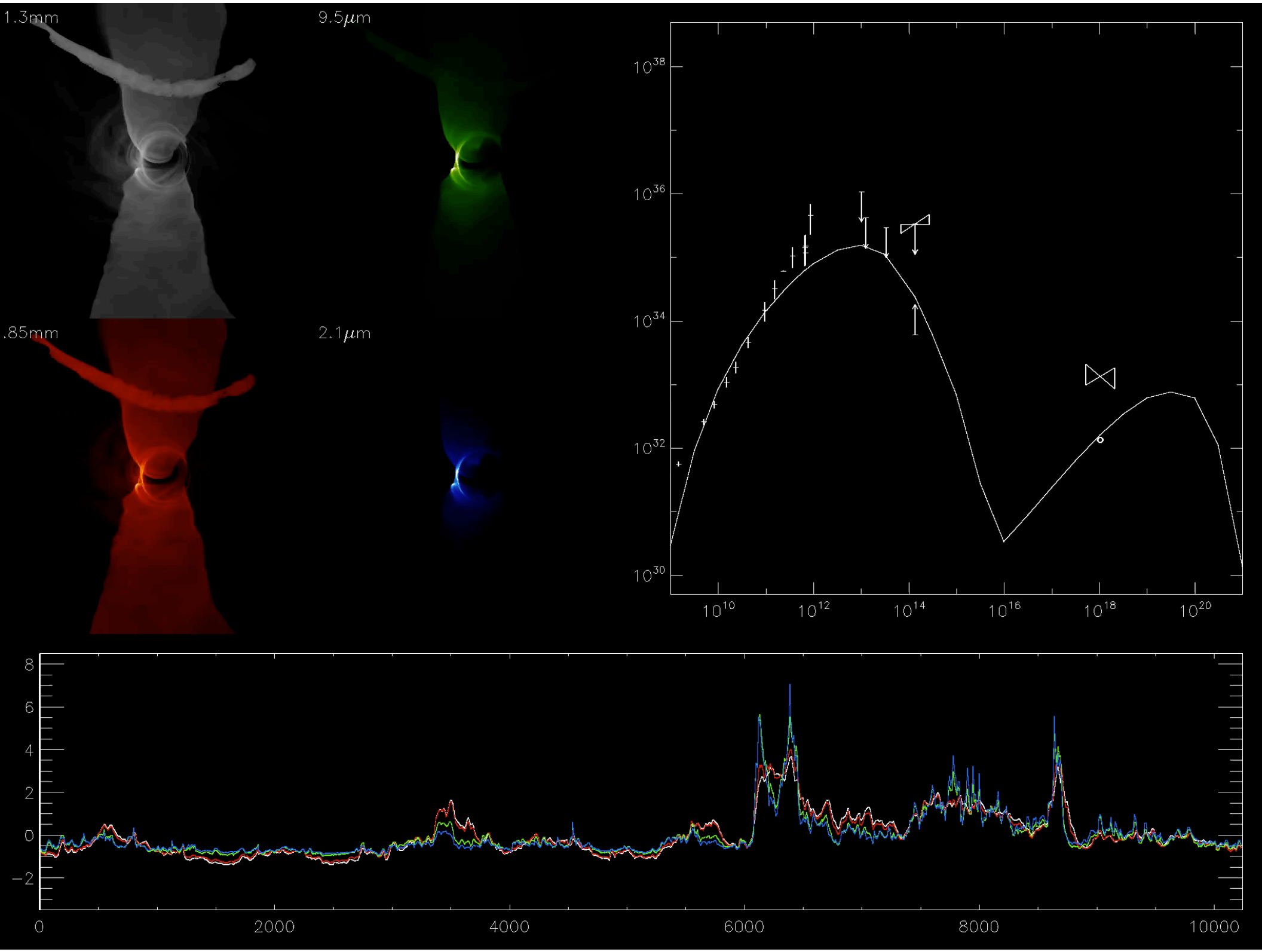
	A	C	D	F	G	H
1	Authors	Code Name	Source	Numerics	Metrics	Technologies
3	Bronzwaer, Davelaar, Younsi, Mościbrodzka, Falcke	raptor	Private	RK4	Arbitrary	CPU/GPU
4	Chan, Ozel , Psaltis, & Medeiros (2017)	gray2	https://github.com/cha	RK4	Kerr/KS	OpenCL
5	Pu, Yun, Younsi, & Yoon (2016)	odyssey	https://github.com/hu	RK5	Kerr/BL	CUDA/C++
6	Dexter (2016)	grtrans	https://www.github.co	Elliptic integrals	Kerr/BL	OpenMP/F90
7	Chen, Kantowski, Dai, Baron, & Maddumage (2015)	kertap	https://bitbucket.org/b	RK45	Kerr/BL	Matlab + Python
8	James, von Tunzelmann, Franklin, Thorne (2015)	dngr	Private		Kerr/BL	Mantra's SL
9	Yang & Wang (2014)	ynogkm	http://cdsarc.u-strasb	Elliptic Integrals	Kerr/BL	F95
10	Chan, Psaltis, & Ozel (2013)	gray	https://github.com/cha	RK4	Kerr/BL	CUDA/C
11	Schnittman & Krolik (2013)	pandurata	Private	CK5	Kerr/BL	
12	Yang & Wang (2013)	ynogk	http://www1.ynao.ac.	Elliptic integrals	Kerr/BL	F95
13	Baubock, Psaltis, Ozel, & Johannsen (2012)	ray	Private	RK4	(Kerr+Qpole)/BL	C
14	Psaltis & Johannsen (2012)				QuasiKerr/BL	
15	Shcherbakov & Huang (2011)	astroray	http://astroman.org/c	RK2	Kerr/BL	C/C++
16	Vincent, Paumard, Gourgoulhon, & Perrin (2011)	gyoto	http://gyoto.obspm.fr/		Kerr/BL + Numeric	C++
17	Dolence, Gammie, Moscibrodzka, & Leung (2009)	grmonty	http://rainman.astro.il	RK4	Kerr/BL	C
18	Dexter & Agol (2009)	geokerr	http://faculty.washingt	Elliptic integrals	Kerr/BL	F77
19	Broderick & Blandford (2004)	vrt2	Private		Kerr/?	
20	Broderick & Blandford (2003)					
21	Miller & Lamb (1998)		Private		Schwarzschild	
22	Reynolds et al ?		Private			
23	Karas, Vokrouhlicky, & Polnarev (1992)		Private	RK45	Kerr/BL	F77

Making Predictions for Black Holes

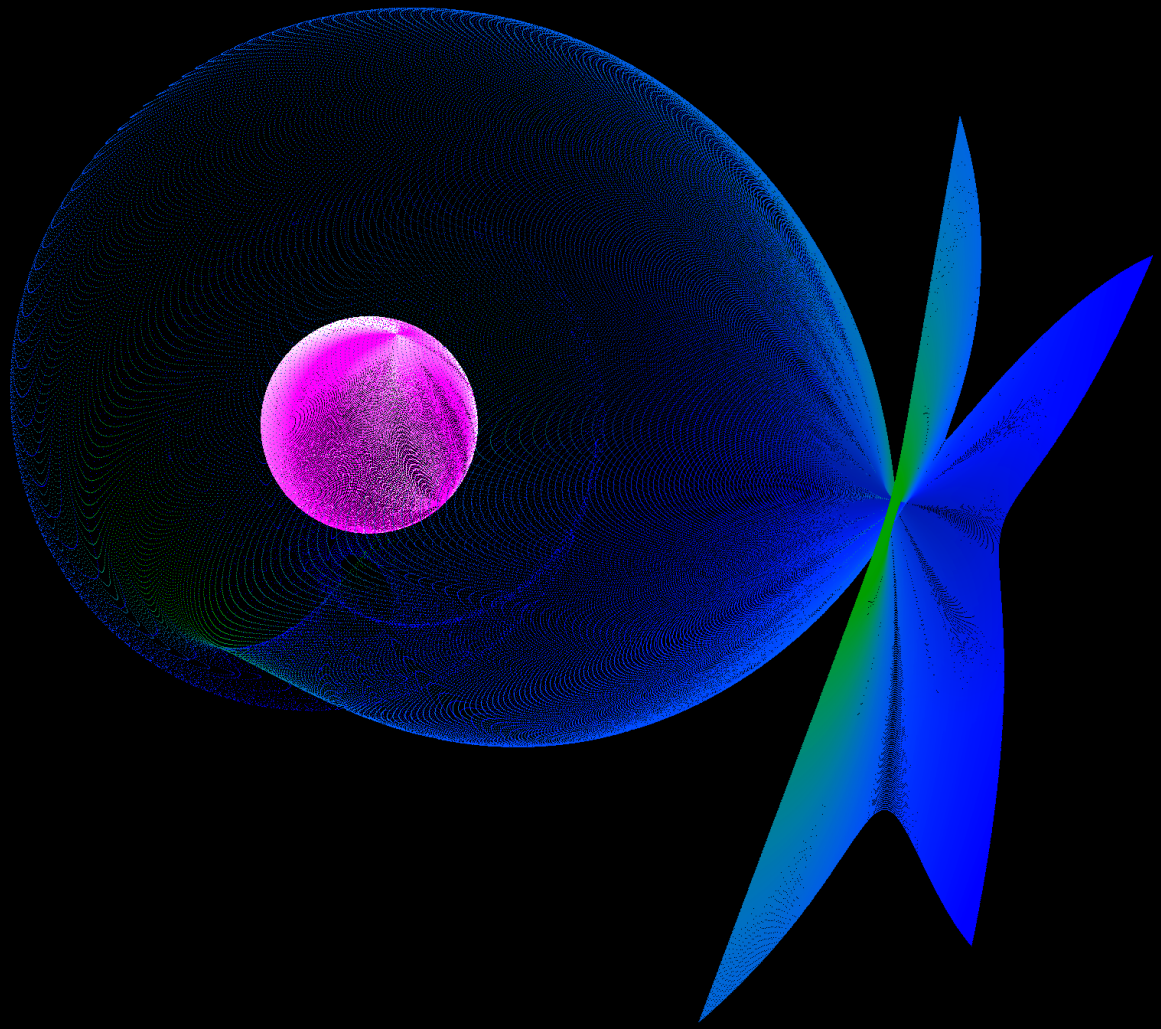


Black Hole PIRE

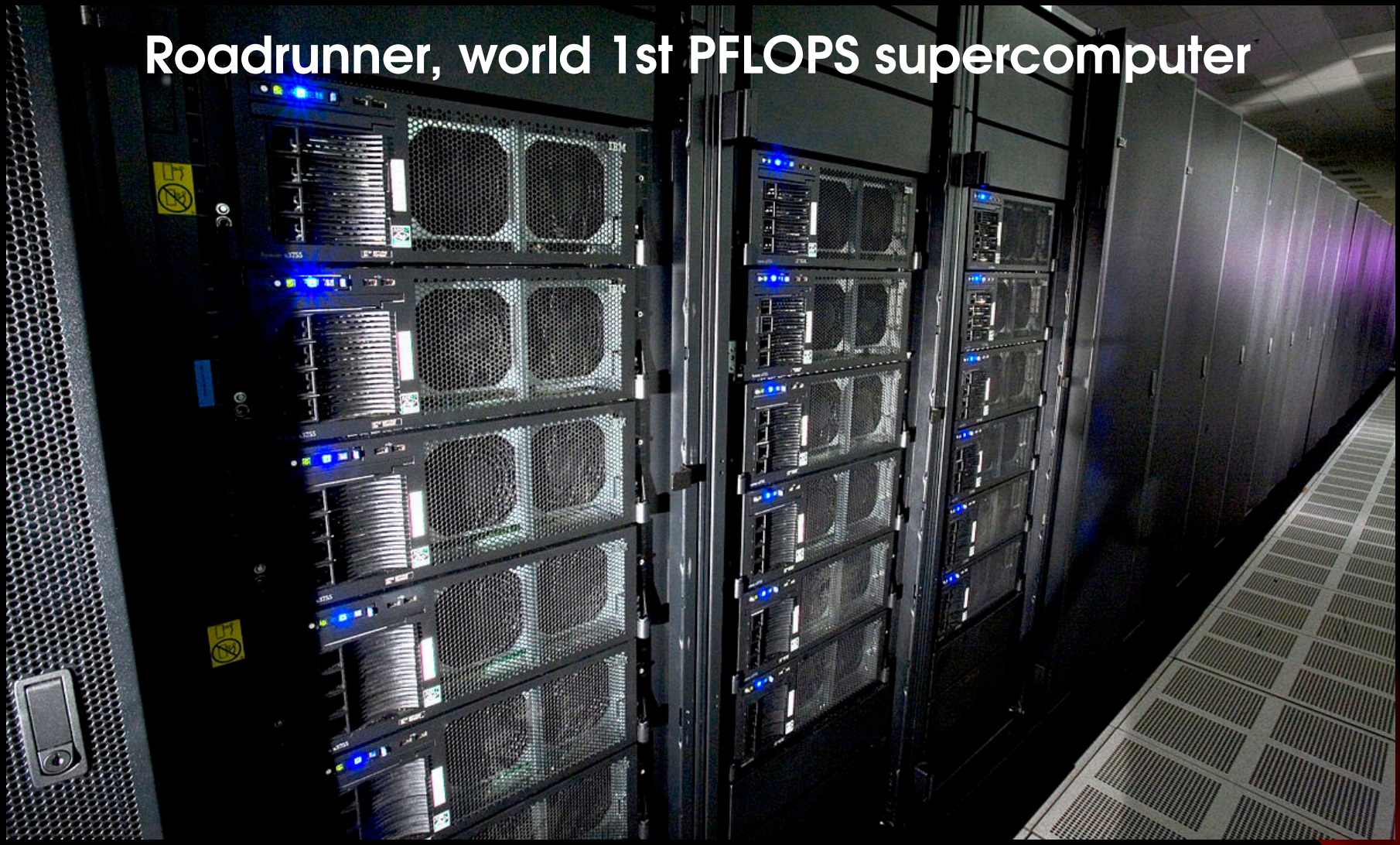

Jet
Disk
Photon Ring


1.3mm
9.5μm
.85mm
2.1μm
10^{38}
10^{36}
10^{34}
10^{32}
10^{30}
10^{10}
10^{12}
10^{14}
10^{16}
10^{18}
10^{20}
8
6
4
2
0
−2
0
2000
4000
6000
8000
10000

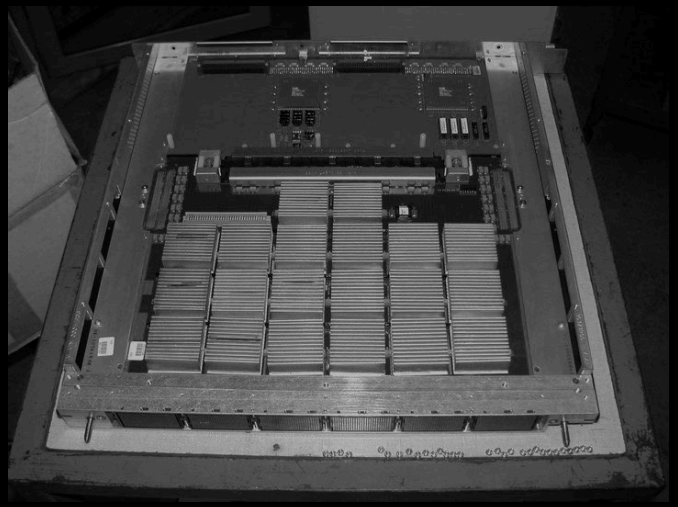
Portable & Adaptive Performance

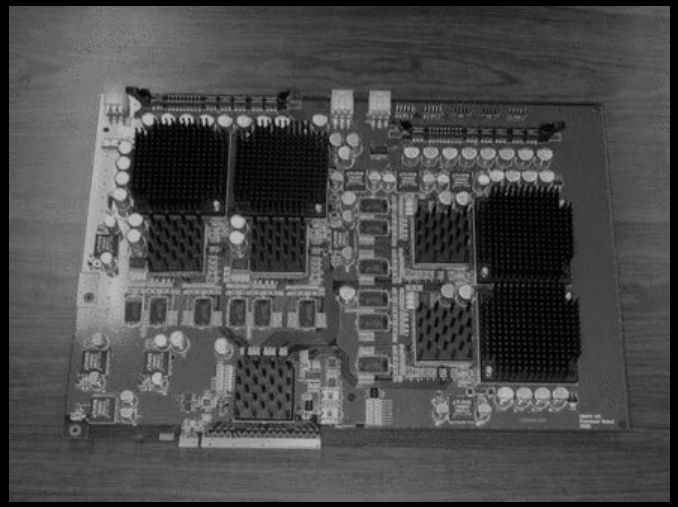


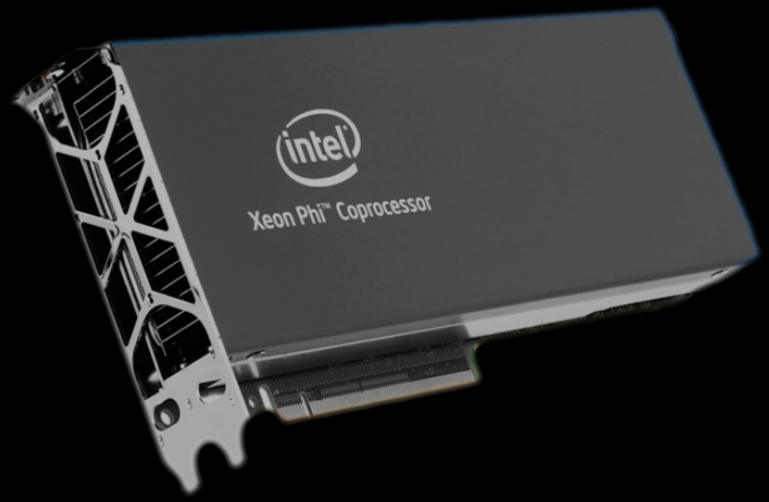
Energy Considerations

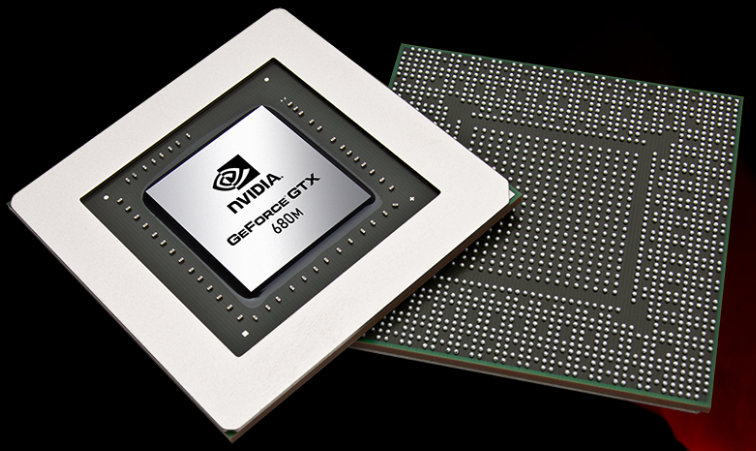


Economy Considerations

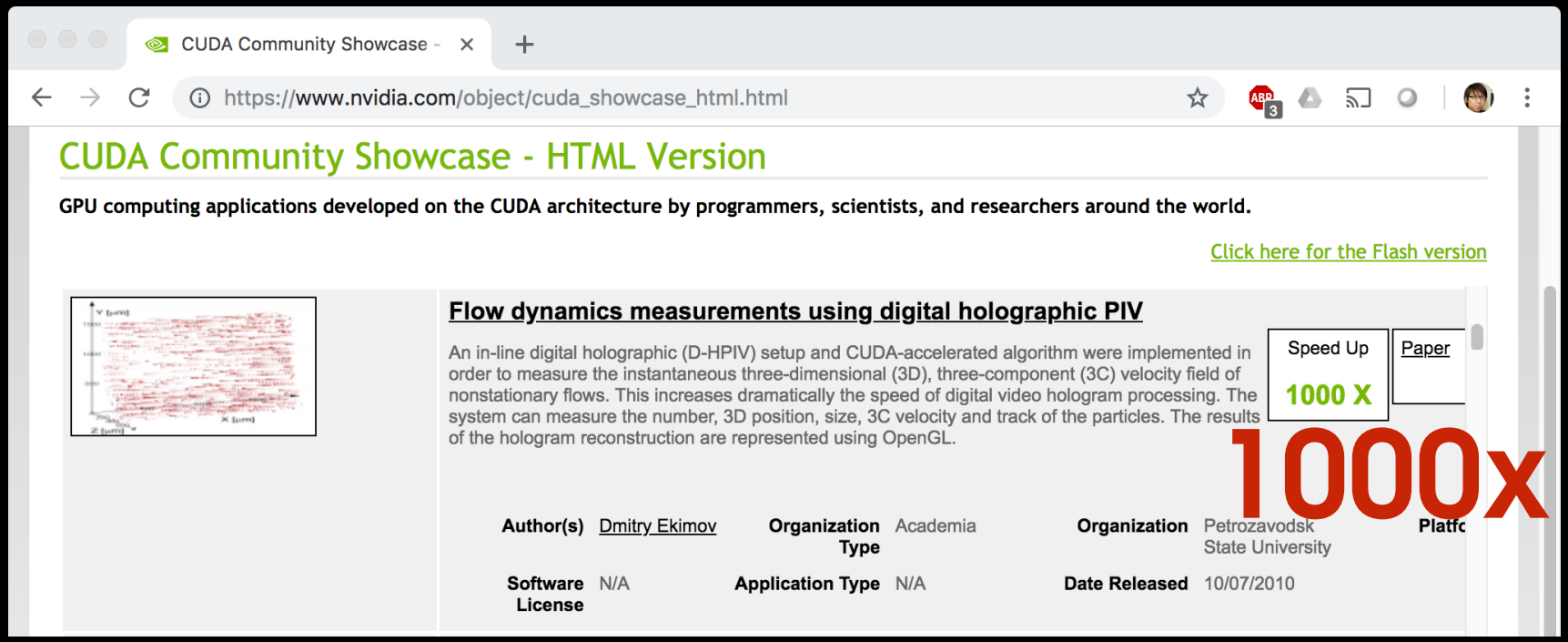








Software Considerations



- Mostly likely because of suboptimal CPU code!
- Learning GPU programming makes you a better CPU programmer

Software Carpentry

ssh-key and "~/.ssh/config"

- Local: `ssh-keygen -f ~/.ssh/id_rsa_pire`
- Local: `scp ~/.ssh/id_rsa_pire.pub user@[IP]:~`
  Remote: `cat id_rsa_pire.pub >> ~/.ssh/authorized_keys`
- `Or: ssh-copy-id -i ~/.ssh/id_rsa_pire.pub user@[IP]`
- Add the following to "~/.ssh/config"

```
Host pire
  User user
  Hostname [IP]
  IdentityFile /Users/ckchan/.ssh/id_rsa_eht
  ForwardX11 yes
  ...
```

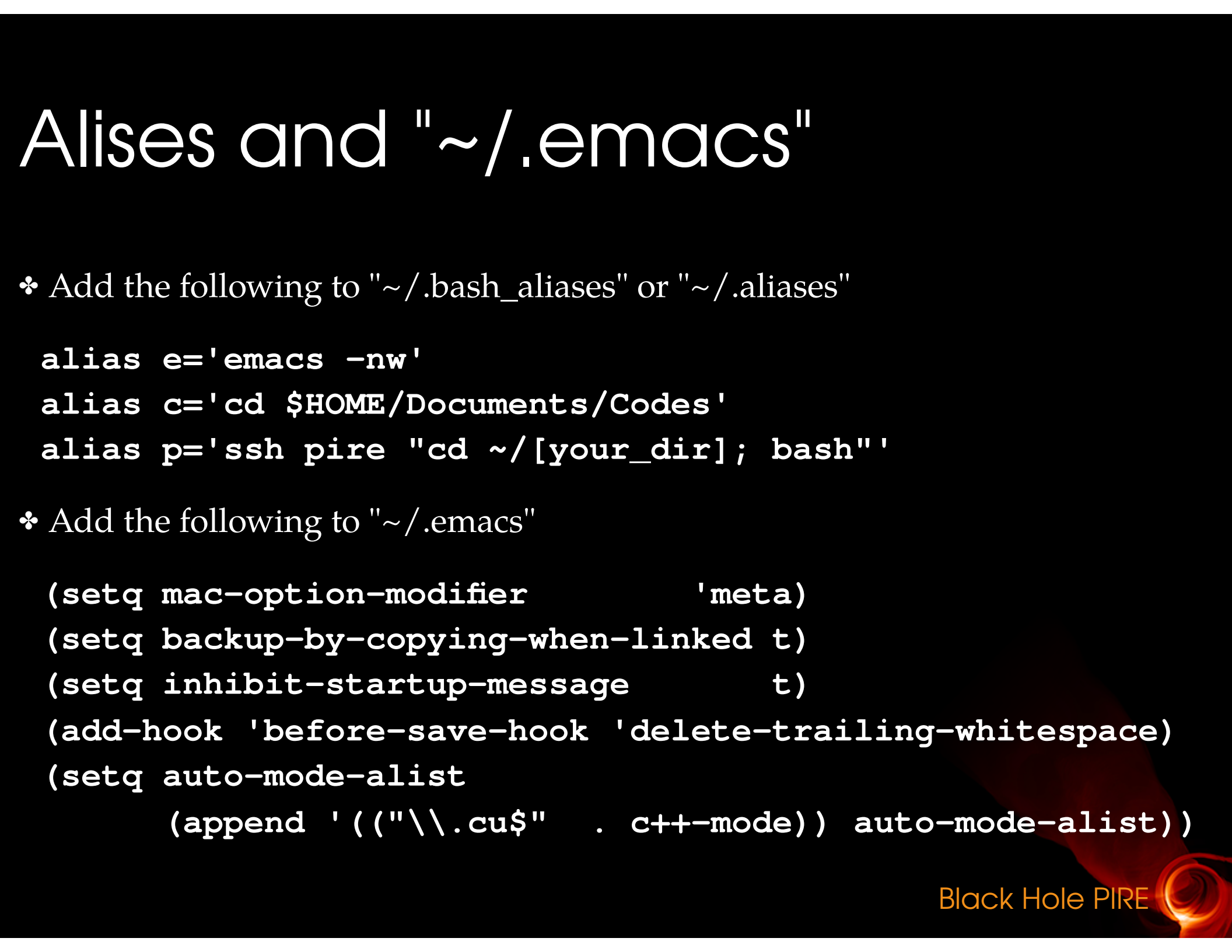

Alises and "~/.emacs"

- Add the following to "~/.bash_aliases" or "~/.aliases"

```
alias e='emacs -nw'
alias c='cd $HOME/Documents/Codes'
alias p='ssh pire "cd ~/[your_dir]; bash"'
```

- Add the following to "~/.emacs"

```
(setq mac-option-modifier          'meta)
(setq backup-by-copying-when-linked t)
(setq inhibit-startup-message       t)
(add-hook 'before-save-hook 'delete-trailing-whitespace)
(setq auto-mode-alist
      (append '(("\\.cu$"   . c++-mode)) auto-mode-alist))
```

~/.gitconfig, .gitignore, and gitk

- Git setup:

```
$ git config --global user.name "Jon Doe"
$ git config --global user.email jondoe@arizona.edu
$ git config --global core.editor emacs
```

  or edit ~/.gitconfig directly

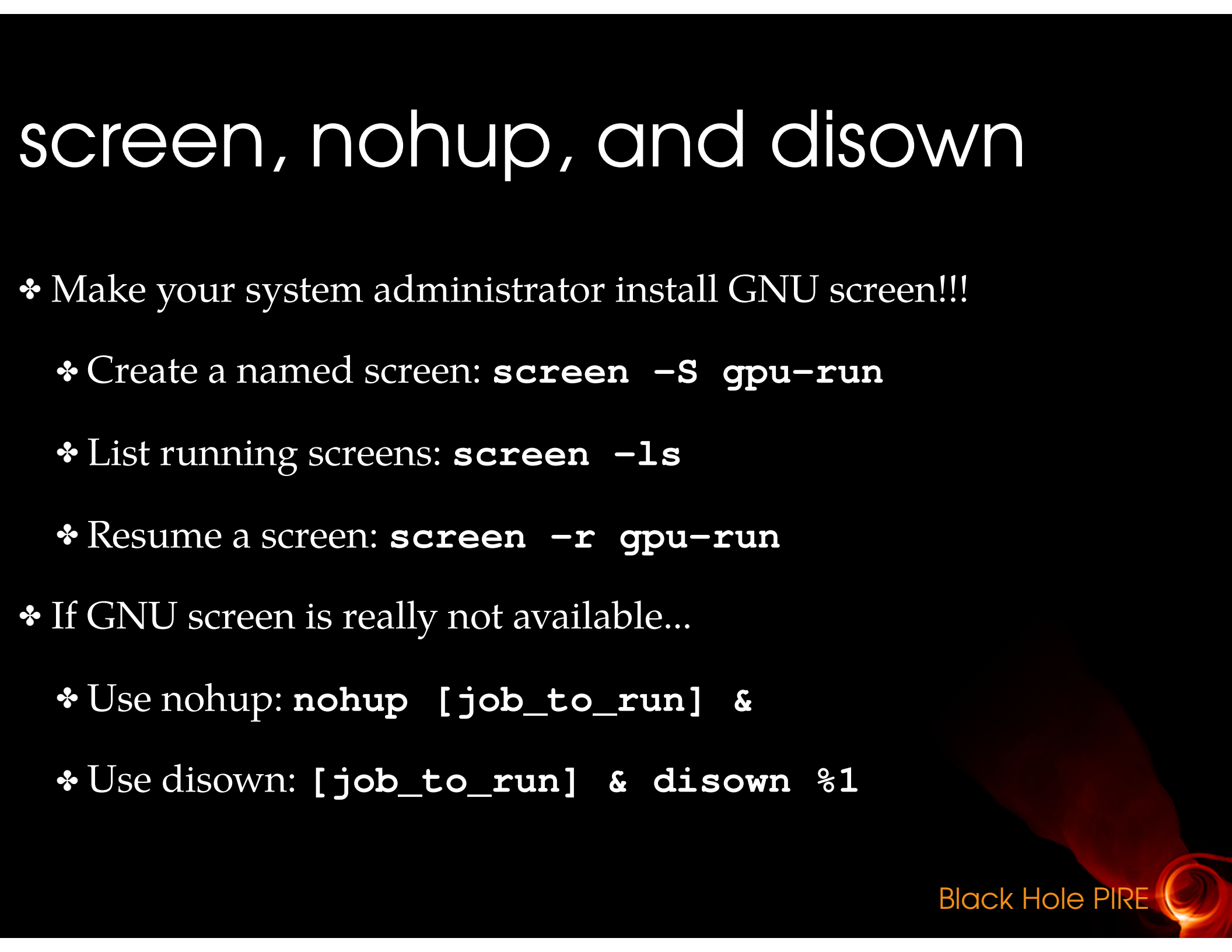
- To ignore most of the files, use the following in .gitignore:

```
*
!*/
```

- Pick a git visualizer, e.g., gitk and tig

screen, nohup, and disown

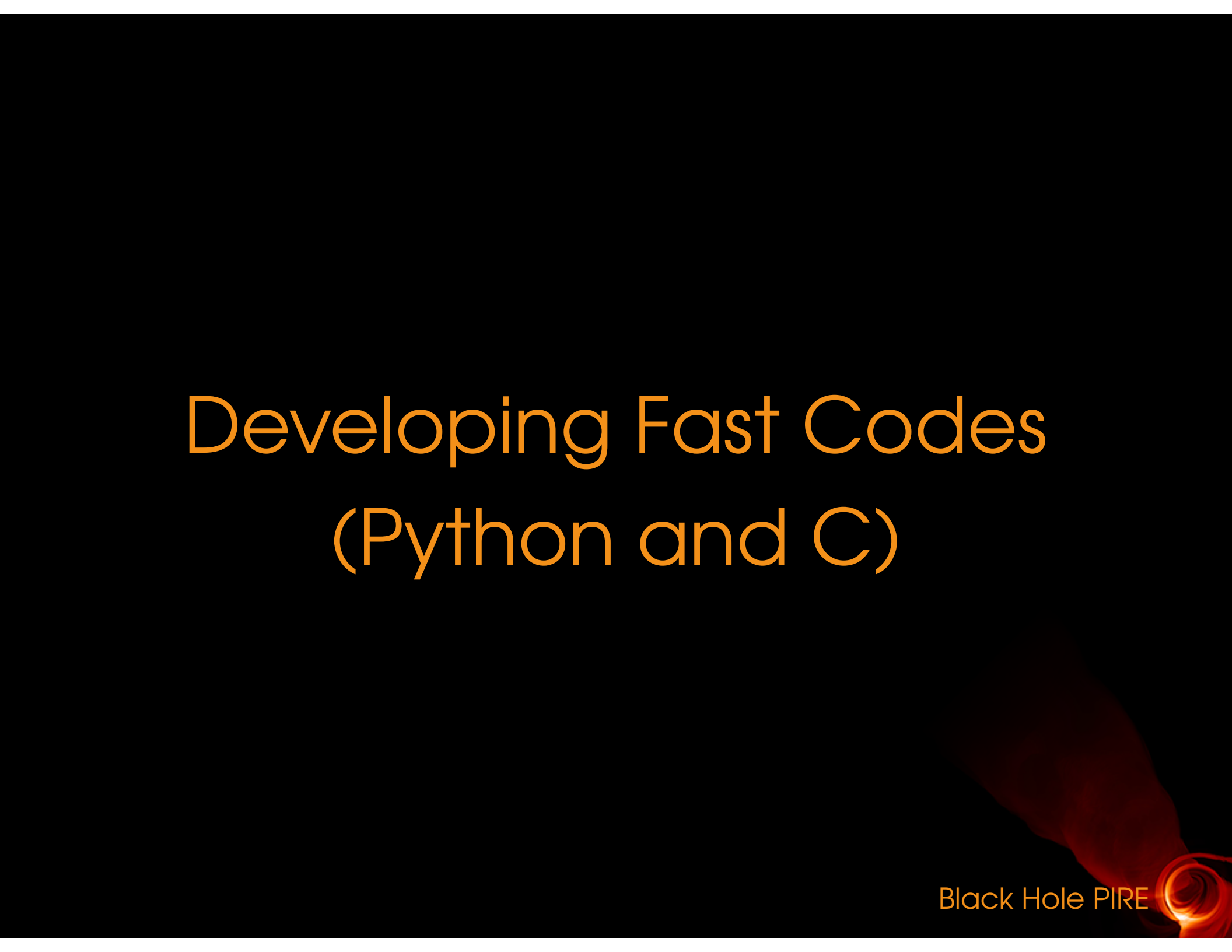
- Make your system administrator install GNU screen!!!
  - Create a named screen: **`screen -S gpu-run`**
  - List running screens: **`screen -ls`**
  - Resume a screen: **`screen -r gpu-run`**
- If GNU screen is really not available...
  - Use nohup: **`nohup [job_to_run] &`**
  - Use disown: **`[job_to_run] & disown %1`**



Good Resources

- For C (in additional to K&R):
  - Read good source codes: coreutils, Linux kernel, fftw, ...
  - http://c-faq.com, e.g., http://c-faq.com/aryptr/ptrtoarray.html
  - https://lwn.net, e.g., https://lwn.net/Articles/336224
- For Python:
  - Read good source codes: flask, werkzeug, request, ...
  - Book: Effective Python: 59 Specific Ways to Write Better Python
  - https://www.kaggle.com/kernels

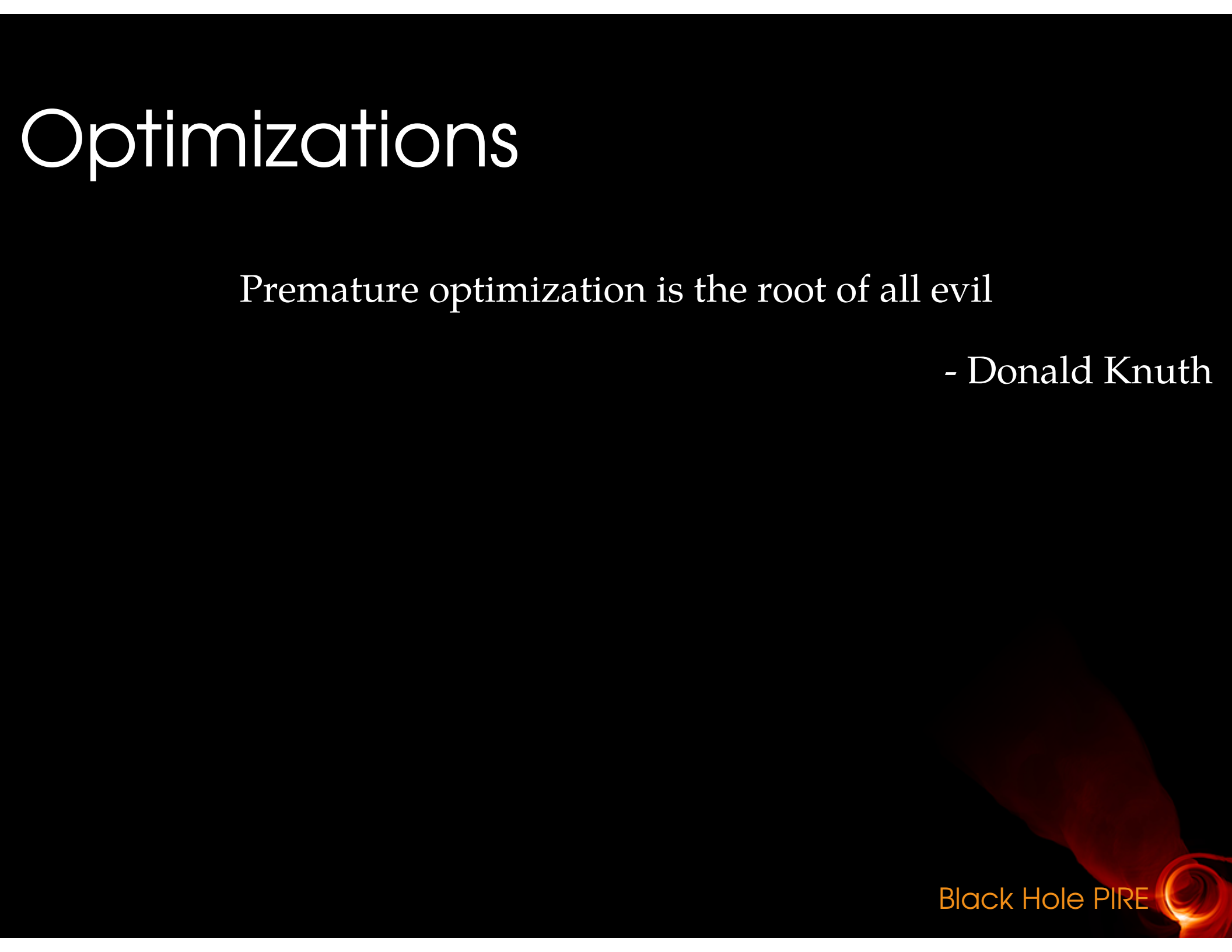
Developing Fast Codes (Python and C)



Optimizations

Premature optimization is the root of all evil

- Donald Knuth



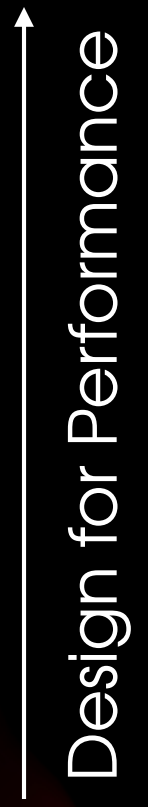
Optimizations

Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.Yet we should not pass up our opportunities in that critical 3%.

- Donald Knuth

Optimizations

- Numerical algorithm, e.g., RK4, high-order finite difference
- Data structure + algorithm, e.g., domain decomposition, neighborhood finder using ochre
  - There are overlaps, e.g.: Barnes-Hut tree
- Scale out using, e.g., MPI or Charm++
  - Amdahl's law: $S(s) = \frac{1}{1 - p + p/s} \leq \frac{1}{1 - p}$
- Optimize for hardware
  - Measure/profile the performance



Just for fun...

- (From FFTW-2) Which code is faster?

```
a = 0.5 * b;
c = 0.5 * d;
e = 1.0 + a;
f = 1.0 - c;
```

```
a = 0.5 * b;
c =-0.5 * d;
e = 1.0 + a;
f = 1.0 + c;
```

Just for fun...

- (From FFTW-2) Which code is faster?

```
a = 0.5 * b;          a = 0.5 * b;
c = 0.5 * d;          c =-0.5 * d;
e = 1.0 + a;          e = 1.0 + a;
f = 1.0 - c;          f = 1.0 + c;
```

- The code on the left is faster. Guess why? The constants 0.5 and -0.5 are stored in different memory locations and loaded separately.
- And this is Not an example telling you not to optimize your code. It's an example telling you to profile your code.

Just for fun...

- (Again from FFTW-2) Array padding: (This is a disgusting hack that my programmer's ethic pervents me from releasing to the public). On the IBM RS/6000, for big n, the following **** is *way* faster:
  - Take the input array
  - `inflate' it, that is, produce a bigger array in which every k-th element is unused (say k=512). In other words, insert holes in the input.
  - execute fftw normally (properly hacked to keep the holes into account)
  - compact the array.

With this hack, FFTW is sensibly faster than IBM's ESSL library. Sorry guys, I don't want to be responsible for releasing this monster (this works only on the RS/6000, fortunately).

Optimizations

- Design your code to perform and scale
- Profile your code, identify critical sections
- Optimize the critical sections, profile them, continue