Automation of Inertial Fusion Target Design with Python

Matt Terry and Joseph Koning
Lawrence Livermore National Laboratory

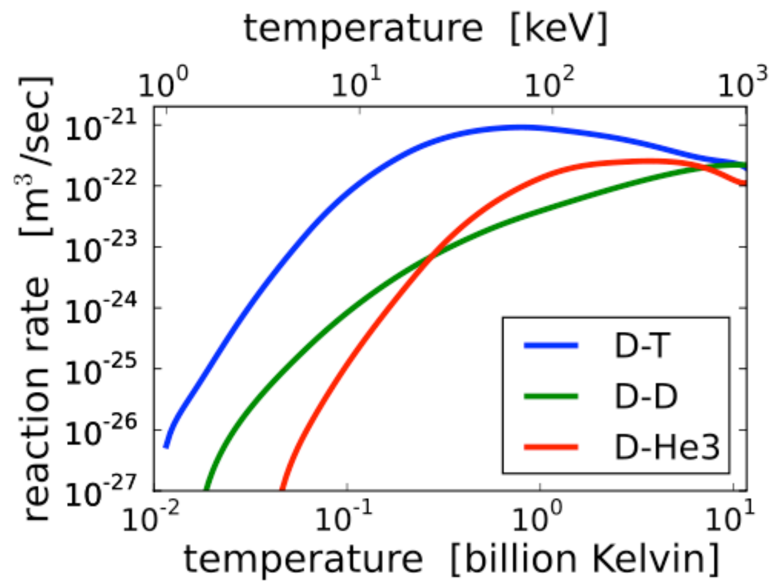
Python for Scientific Computing Conference
Austin, Texas
July 13, 2011

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94551

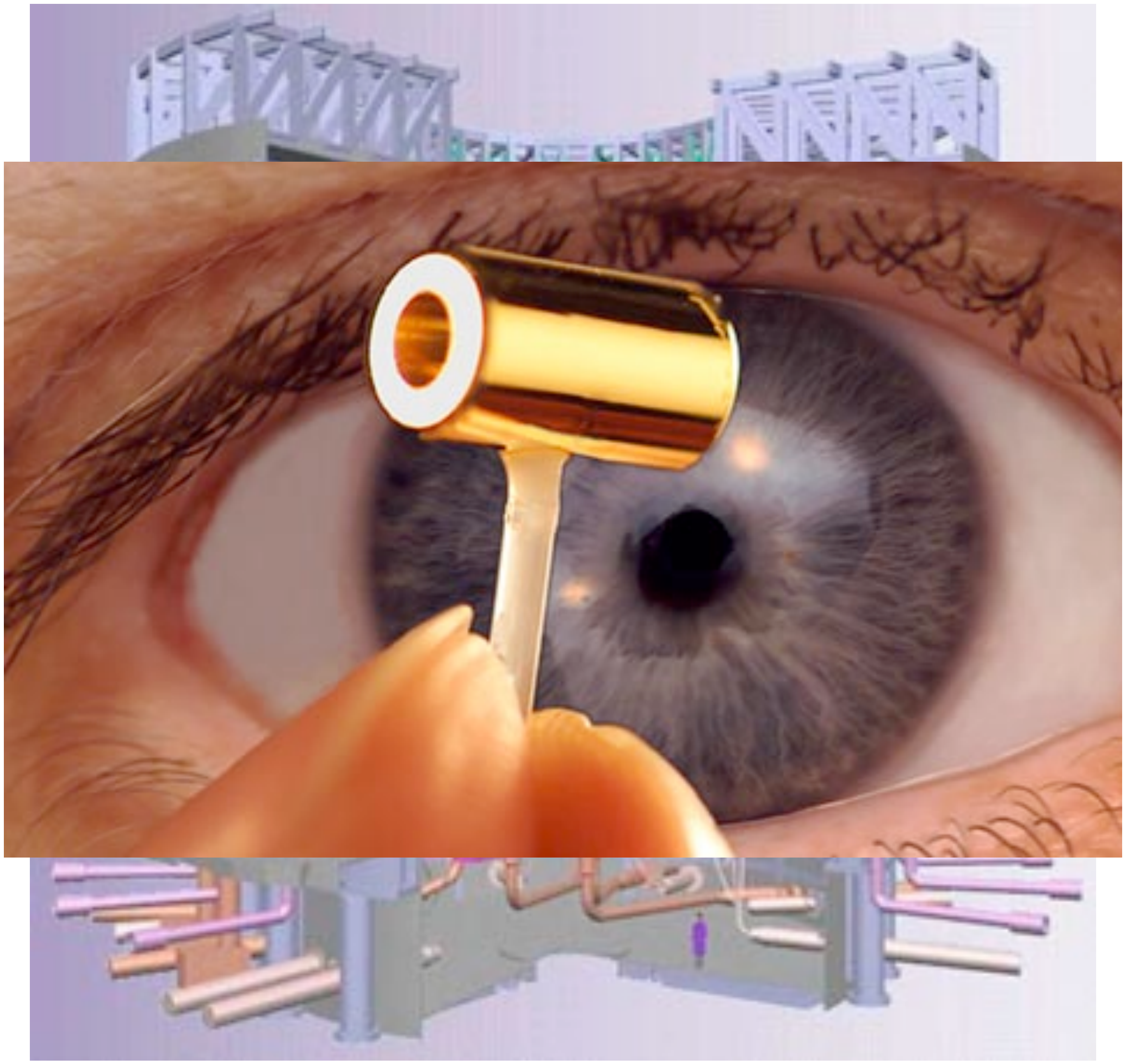
Fusion releases energy by combining small light nuclei into heavier nuclei

- **Thermonuclear fusion requires very high temperatures to overcome the Coulomb barrier**
- **Energy must be confined long enough to react and propagate**
- **Need sufficient density (fuel) to release significant energy**
- **Lawson criteria illustrates required ignition conditions**



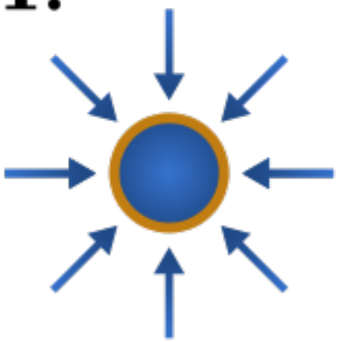
$$n_e T \tau_E > c$$

Go BIG or go FAST



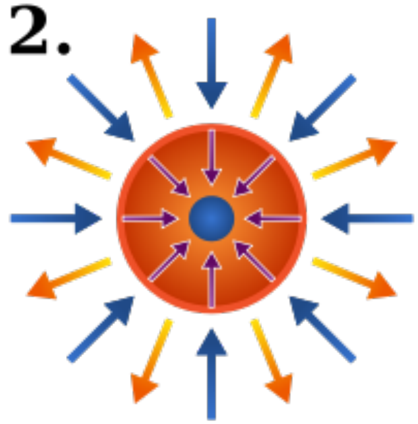
ICF works by compressing DT fuel to high density and relying on the fuel inertial to confine it

1.



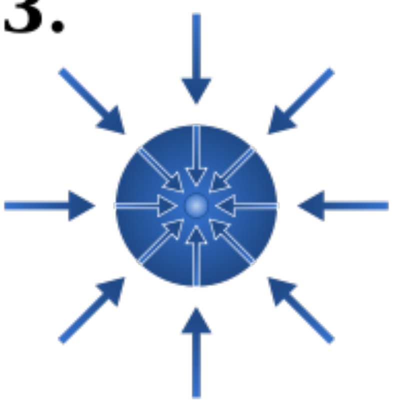
A high intensity driver (x-rays, laser, heavy ion beam, etc) illuminates the surface of a layered spherical pellet

2.



The outer surface of the pellet ablates, compressing DT fuel and driving an imploding spherical rocket

3.

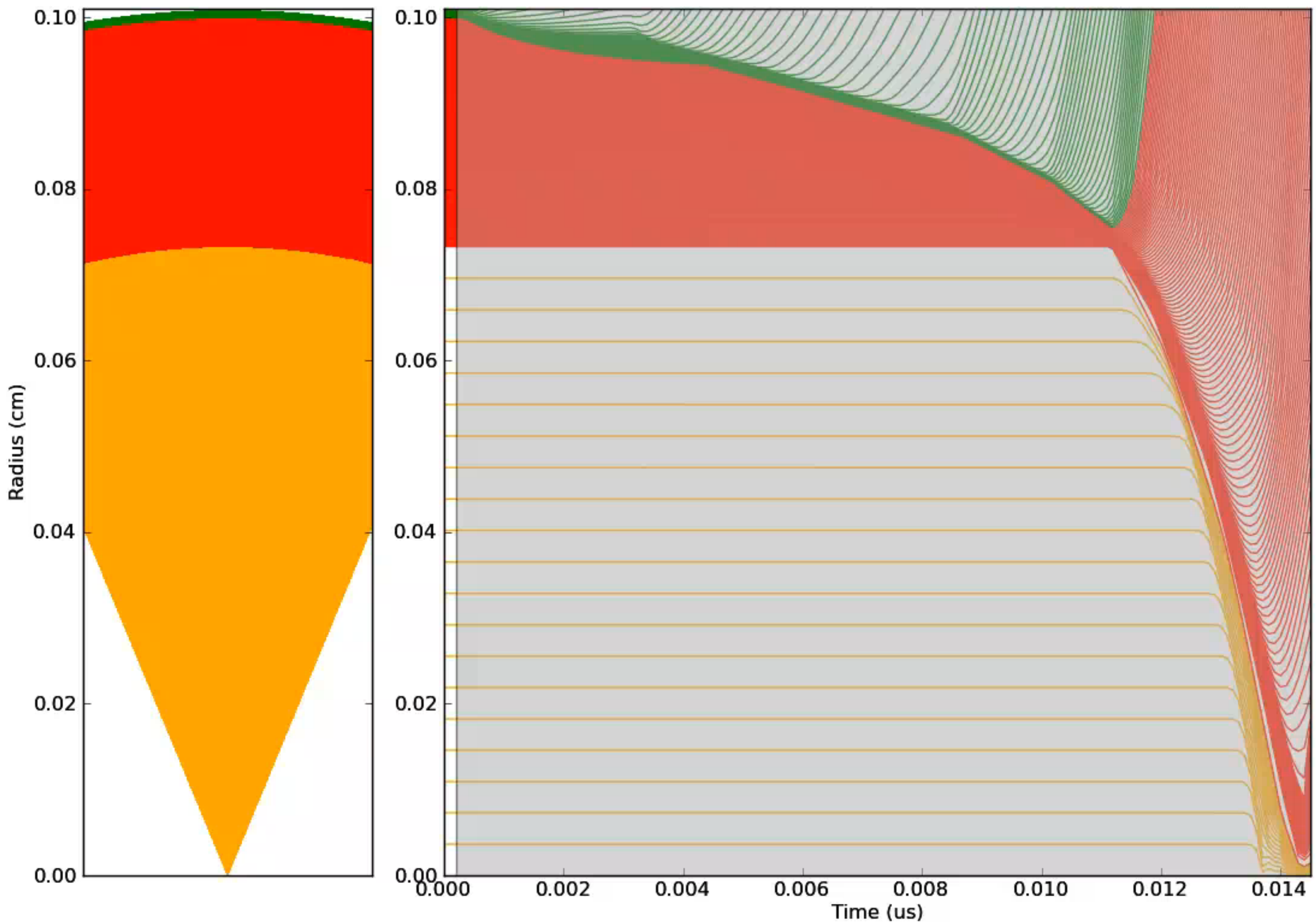


The dense imploding shell stagnates on axis and converts its kinetic energy to thermal energy

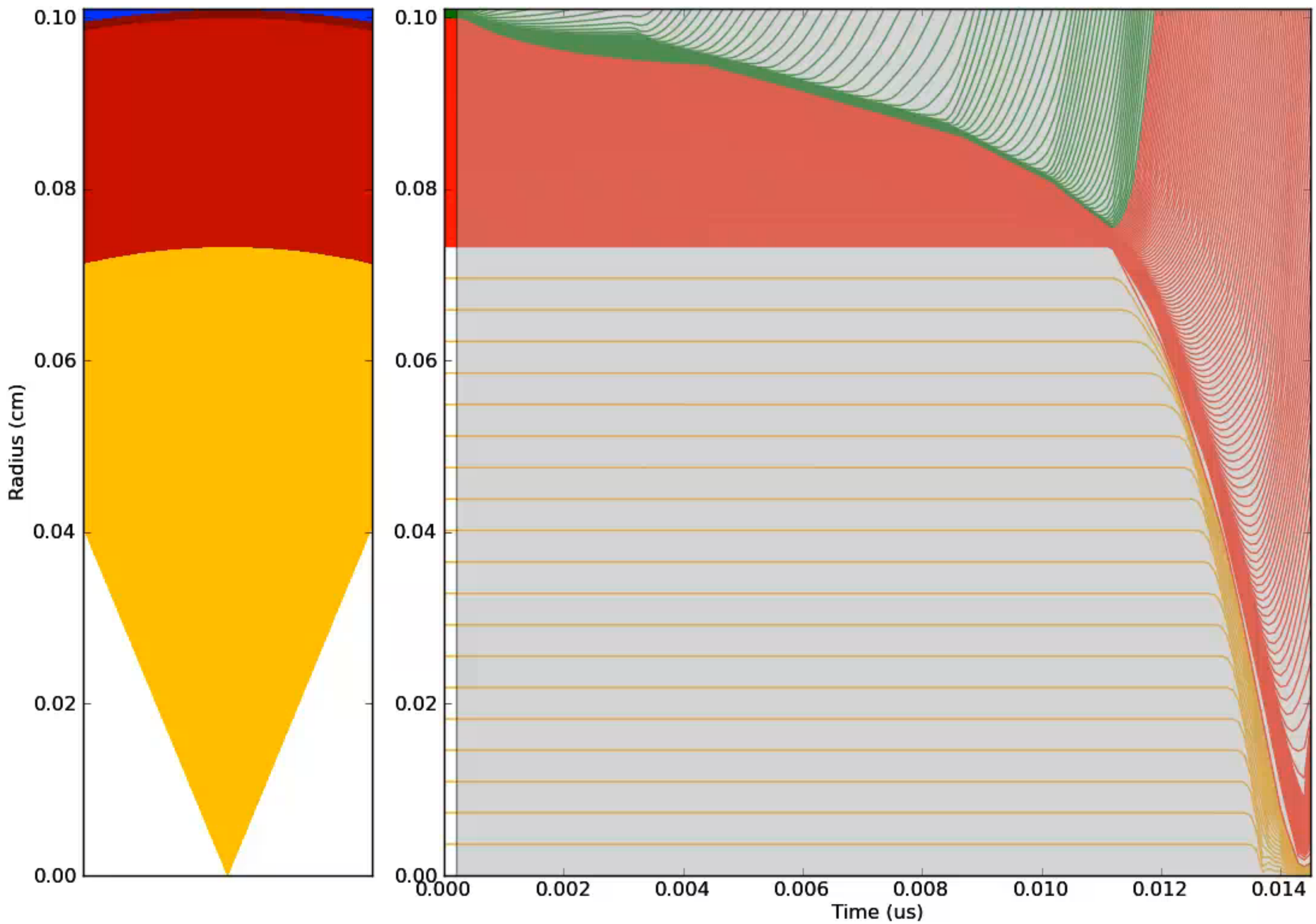
4.

The high stagnation temperature initiations a fusion burn wave. The burn wave propagates faster than the shell can disassemble, releasing large amounts of energy

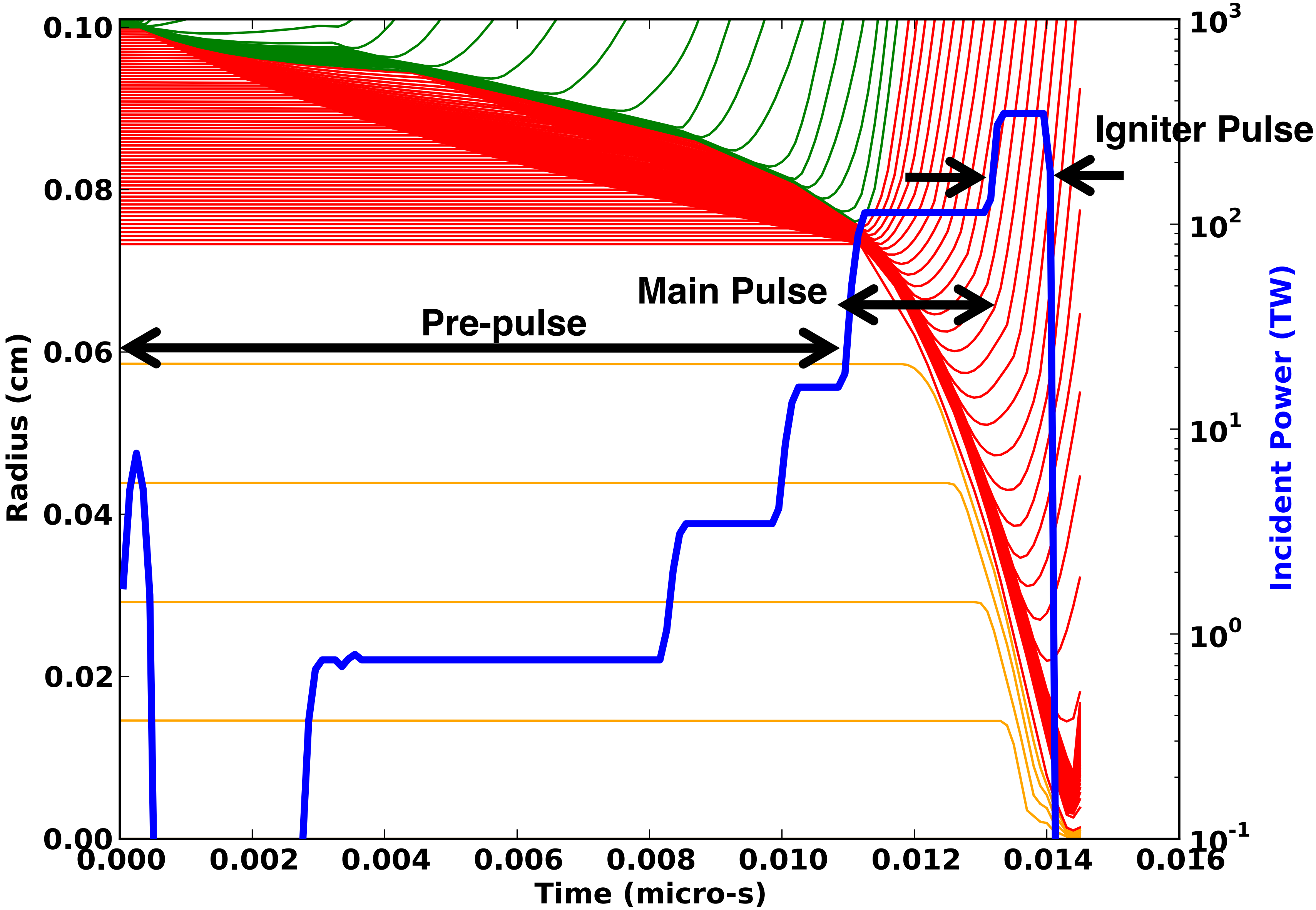
http://en.wikipedia.org/wiki/File:Inertial_confinement_fusion.svg



Radius (cm)
0.10
0.08
0.06
0.04
0.02
0.00
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
Time (us)



Radius (cm)
0.10
0.08
0.06
0.04
0.02
0.00
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
Time (us)



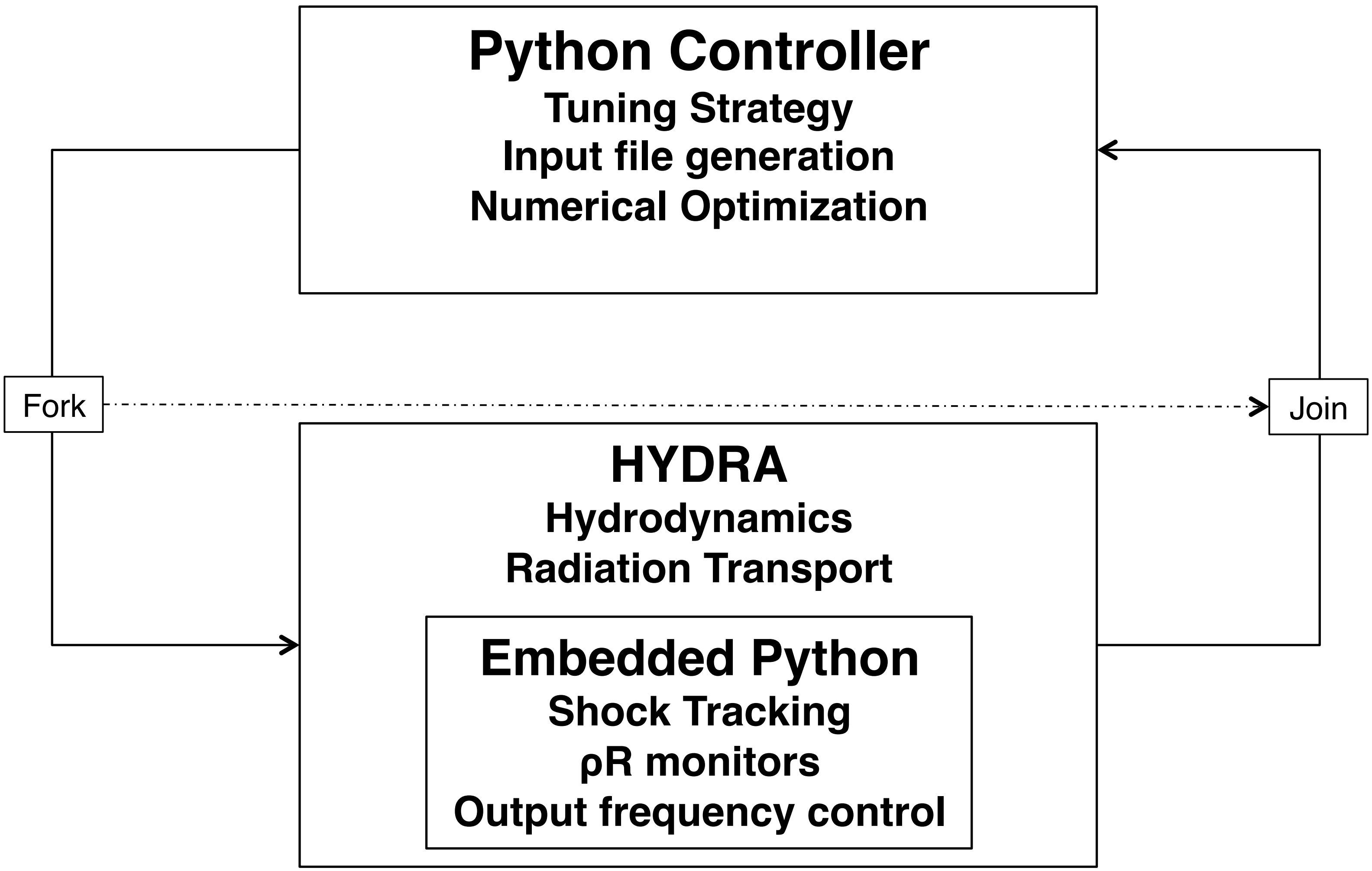
Igniter Pulse
Main Pulse
Pre-pulse
Radius (cm)
Incident Power (TW)
Time (micro-s)
0.10
0.08
0.06
0.04
0.02
0.00
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016
10^3
10^2
10^1
10^0
10^{-1}

Build a tuned pulse by appending optimized the pieces

- **Pulse start times and powers must be adjusted to have the correct behavior**
- **Adjusting (“tuning”) by hand has high latency and is labor intensive**
- **We can construct objective functions that implement the “eye balling” heuristics that people use when manually tuning**
- **Construct a tuned pulse by appending tuned pieces**

```
tuners = 3*[shock_sync] + [max_rhor, max_yield]
segments = [‘shock2’, ‘shock3’, ‘shock4’, ‘main’, ‘ignite’]
laser = Laser()
for tuner, seg in zip(tuners, segments):
    tune_val = tuner(deck, laser, seg)
    laser[seg] = tune_val
```

Autotuner Program flow



Hydra is a massively parallel multi-physics code developed at LLNL

- **50+ users at National Labs and in academia**
- **Use to model high energy density plasmas**
- **In development since 1993**

- **ALE Hydrodynamics**
  - **2D / 3D block structured mesh**
- **Radiation (Photon) transport**
- **Laser beams**
- **Heavy ion beams**
- **Resistive MHD**
- **Ion/Electron conduction**
- **PIC**
- **Atomic physics**
  - **Equation of State**
  - **Opacity**
- **Fusion Burn**

Python is both extended by and embedded in Hydra

- Python is extended through a module called “hydra” which contains functions and objects to manipulate hydra data structures
- Python is then embedded in the Hydra executable
- Bottom line: Hydra makes available a Python interpreter running concurrently in parallel with the main Hydra executable. The two processes are loosely coupled through the “hydra” Python module

Interesting lessons learned

- Python readline cannot be overridden for non-interactive tty's
  - Python is not the primary interpreter
  - Subject to master node, so not even second in line
  - Solution: use custom interactive interpreter

- Saving the state of `__main__` across restarts is tricky
  - Cobble together many methods to collect state
  - Pickle state as a string and add it to restart files
  - Some objects pickle but do not unpickle!

- If your program uses different indexing than Python, resist the urge to emulate your program's indexing in Python

Embedded Python interpreter enables “introspective” programs without major software development effort

- Flexible in code diagnostics
  - High frequency sampling without dumping to disk
  - Watch for shock breakout
  - Watch for peak ρR
- Steer simulation based on gathered information
  - Finer output resolution near interesting features
  - Turn on additional physics based on software triggers
- Don’t need access to the source & no recompile

Auto-tuner needs to generate, run and process many simulations

- Input file generators that wrap a "template" input file
  - Modify simple Hydra variable assignments
  - Delegate complicated input file structures to special purpose objects
  - Inject commands into the input file by overwritings sentinels with `str(python_proxy_of_complicated_thing)`

- Output file wrappers extract data from Hydra binary output files

- Embedded diagnostics using embedded Python interpreter
  - Characteristic (shock) trackers with finish line triggers
  - Change graphics dump frequency based on shock locations
  - ρR monitors

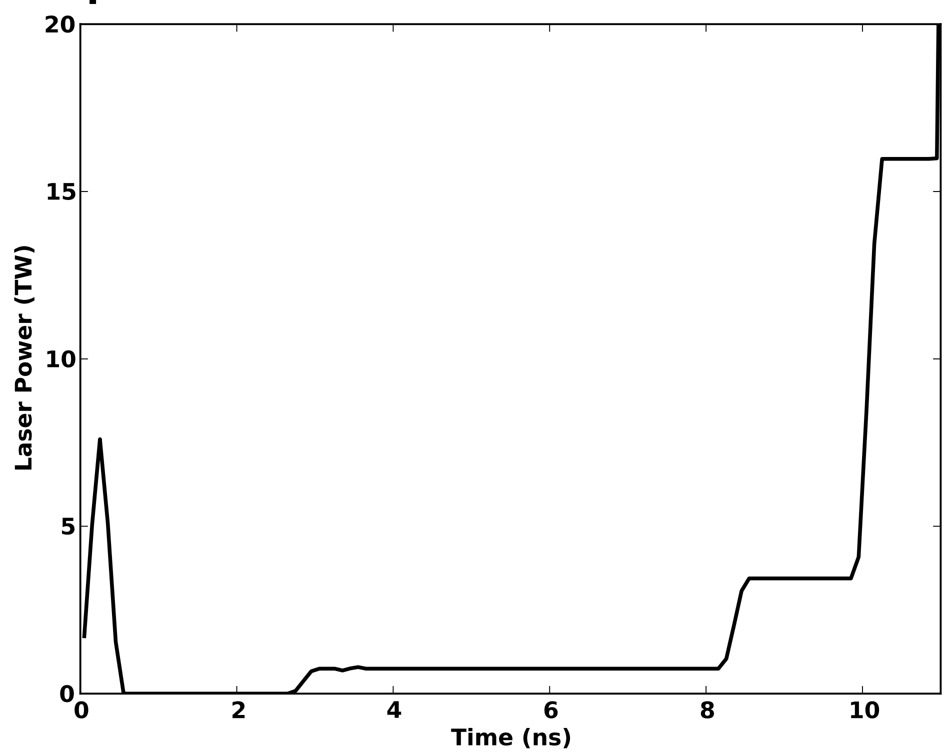
Physics and design rational for pre-pulse

- **Picket launches a decaying 1 MBar shock and acts as a fiducial for synchronizing subsequent shocks**
- **Intensity and shape of picket are set by 2D stability considerations and are assumed as givens for this work (see future work)**
- **Pedestals launch shocks which increase density by ~4x**
  - **Near maximum compression for a planar shock**

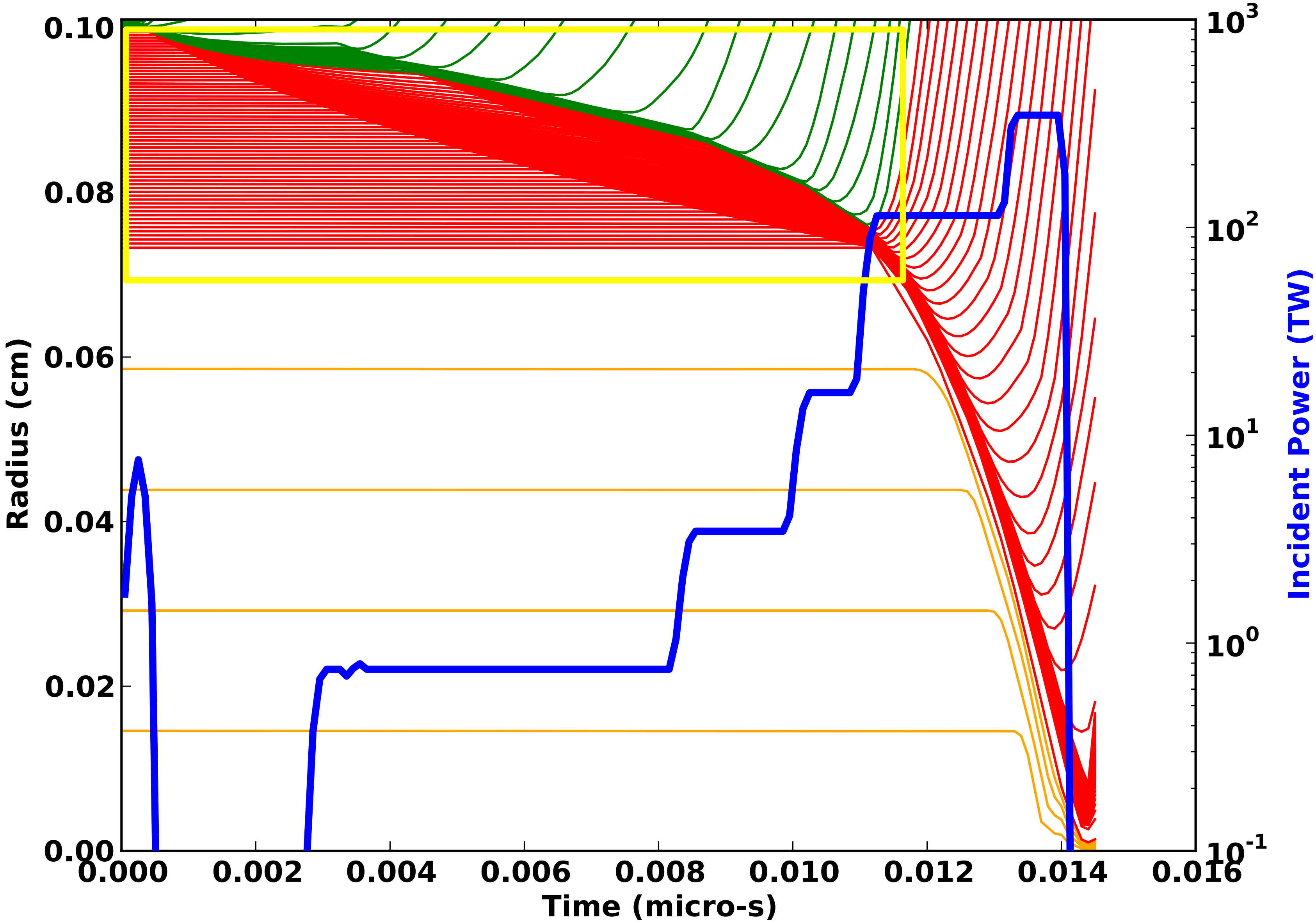
$$\frac{\rho_2}{\rho_1} = \frac{p_2(\gamma+1) + p_1(\gamma-1)}{p_1(\gamma+1) + p_2(\gamma-1)}$$

$$\frac{\alpha_2}{\alpha_1} = \frac{p_2}{p_1}\left(\frac{\rho_1}{\rho_2}\right)^{\gamma}$$

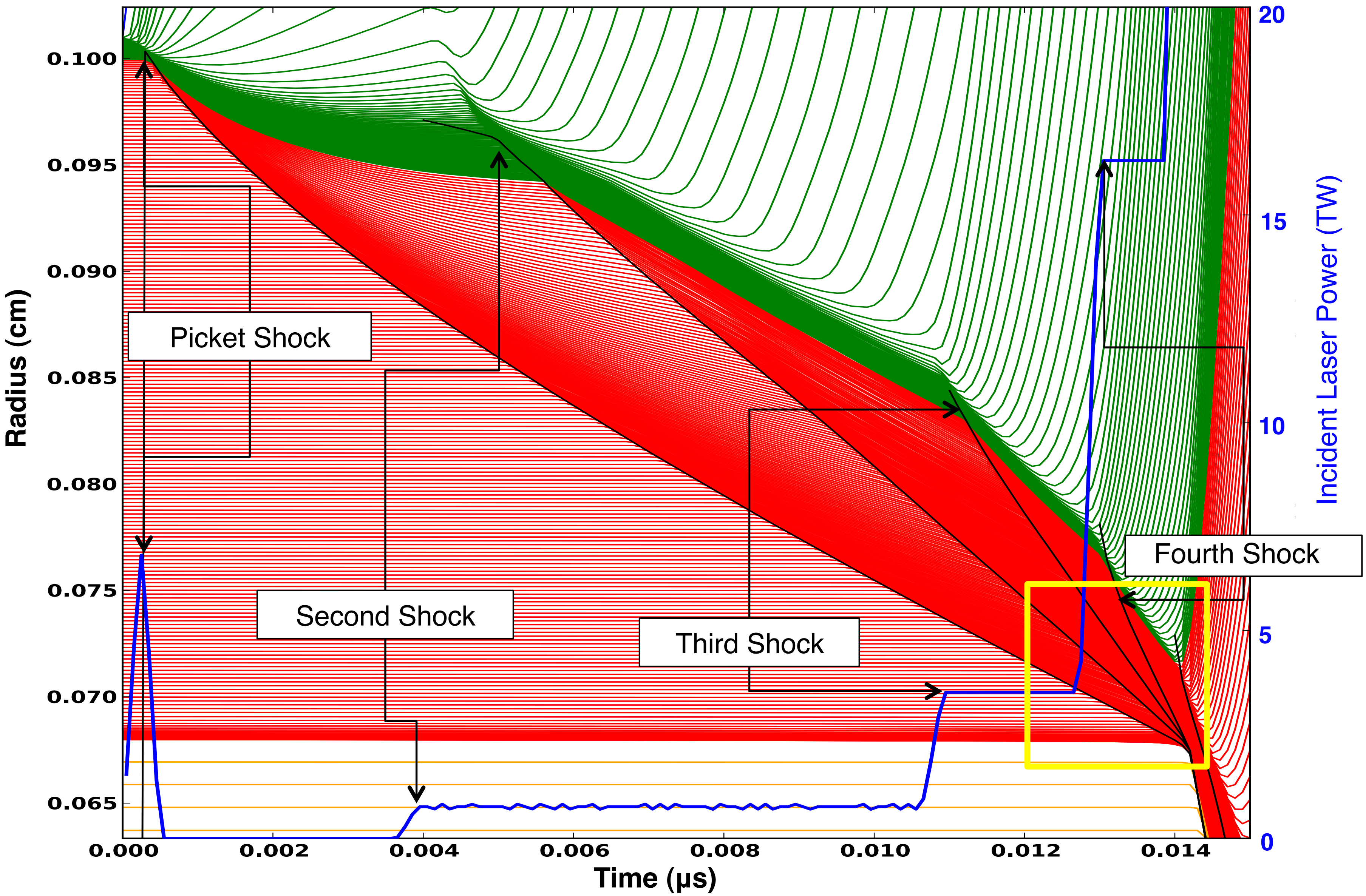
  - **Stronger shocks decrease performance by increasing entropy (drop compressibility)**
- **Maximum fuel compressibility is very important so shocks should coalesce outside of fuel**



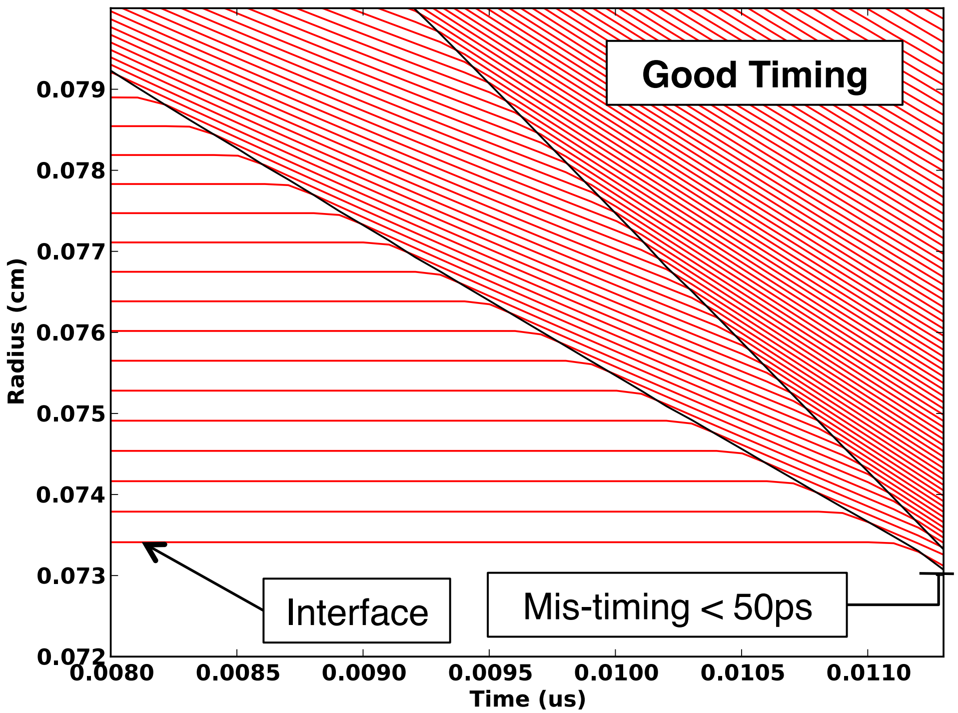
Lets focus on the pre-pulse shocks

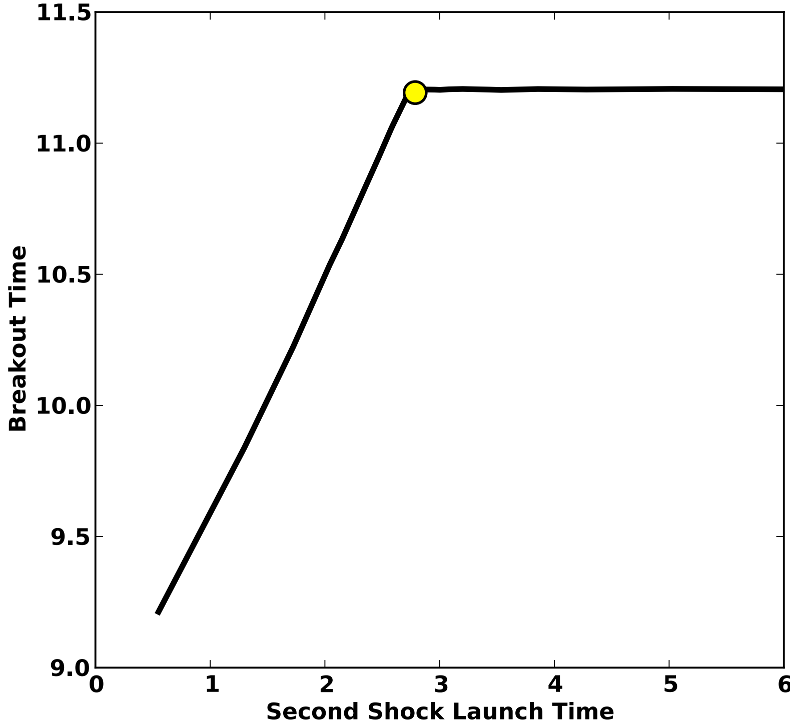


Each pulse segment launches shock which can be unambiguously identified and tracked



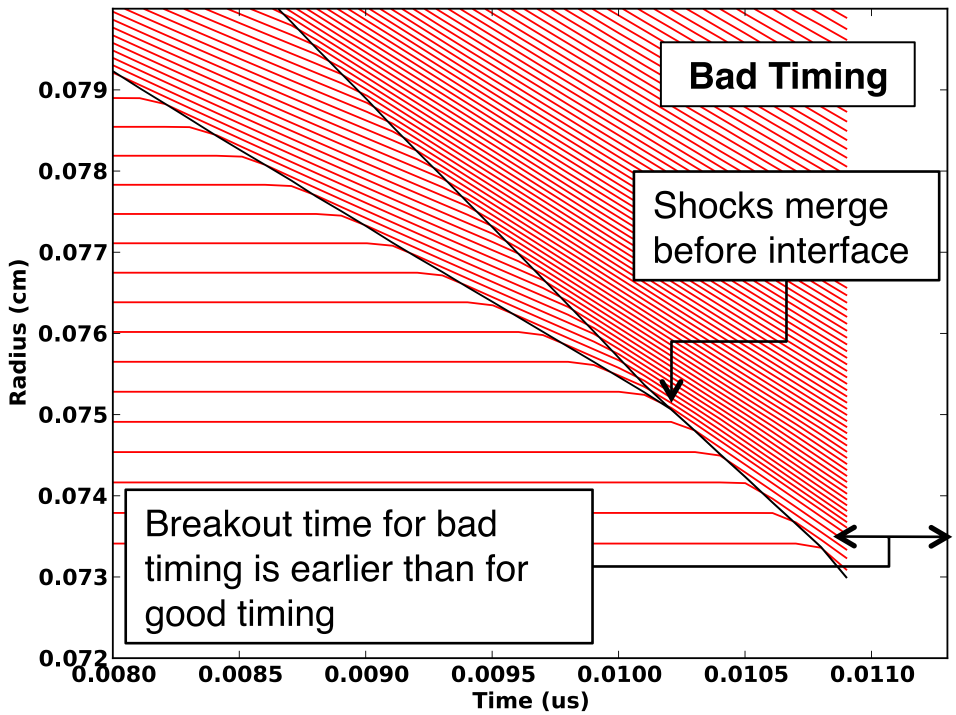
To maintain low fuel entropy, pre-pulse shocks should be timed to coalesce at gas/ice interface within 50 ps of each other

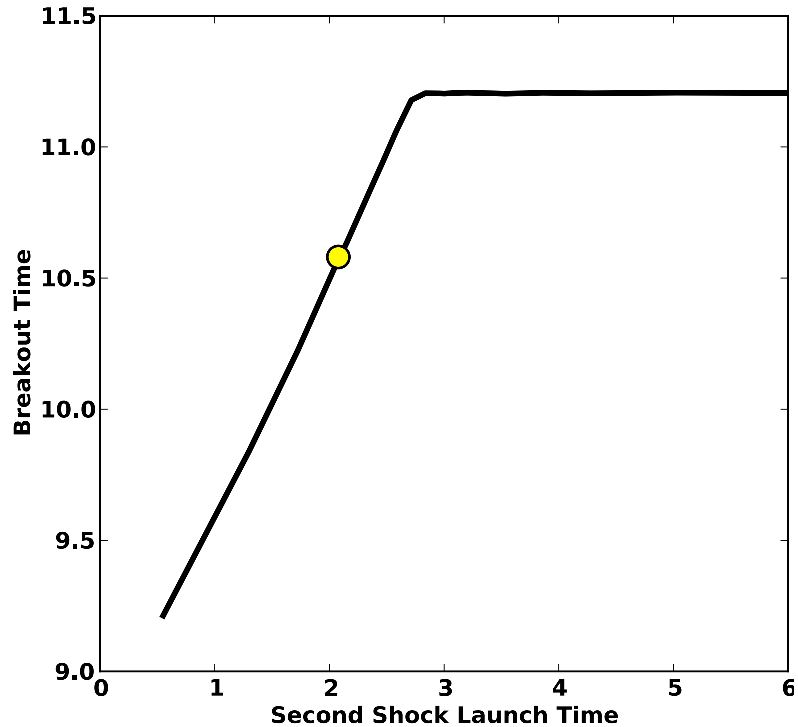




- "Breakout time" is when the first shock crosses the gas/ice interface
- All pre-pulse shocks should coalesce at the gas/ice interface at the same time

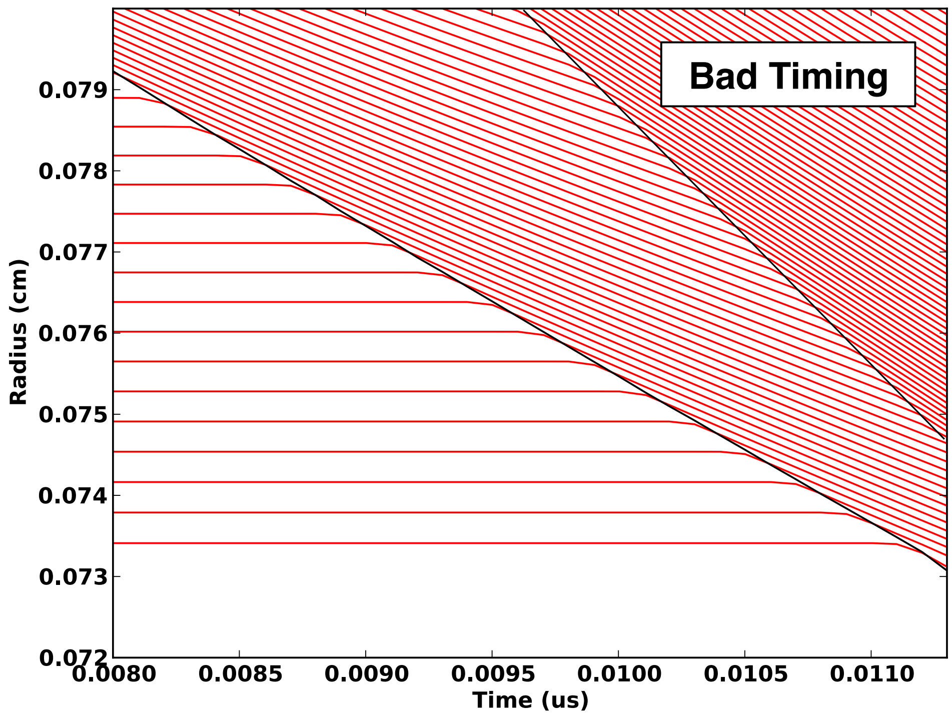
To maintain low fuel entropy, pre-pulse shocks should be timed to coalesce at gas/ice interface within 50 ps of each other

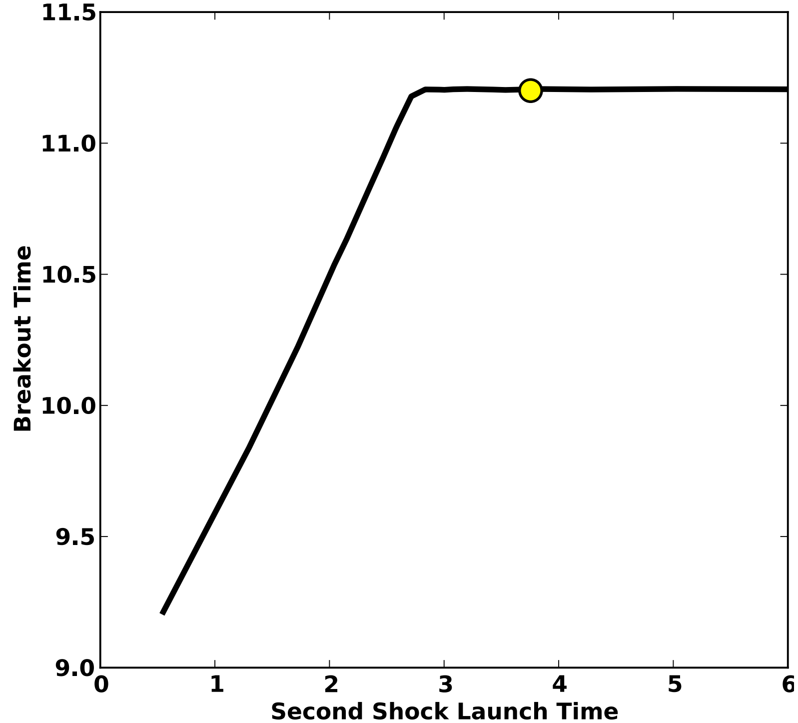




- Shocks coalesce in fuel if second shock launched too soon
- Combined shock is faster than individual shocks
- Breakout time decreases

To maintain low fuel entropy, pre-pulse shocks should be timed to coalesce at gas/ice interface within 50 ps of each other





- If the second shock is launched too late, the first shock "wins"
- Breakout time is unchanged

We construct an objective function which is biased to find the earliest launch with the saturated breakout time

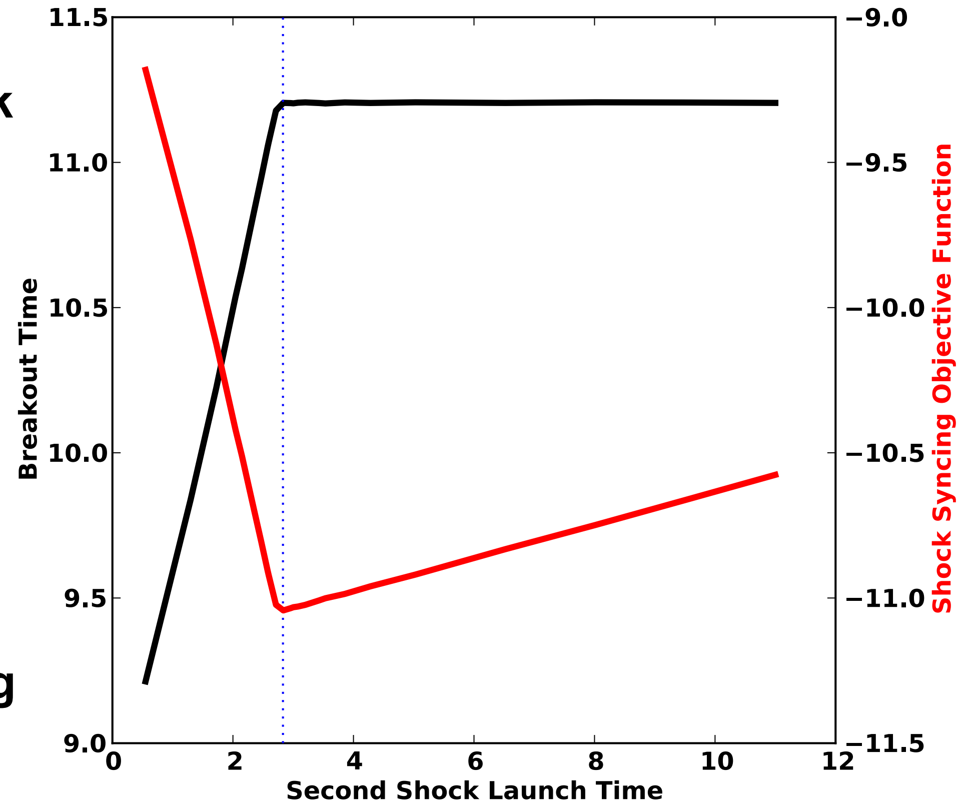
- **A tuned shock is the earliest shock launched that also has the asymptotic breakout time**

$$f_1(t) = b(\infty) - b(t) = -b(t)$$

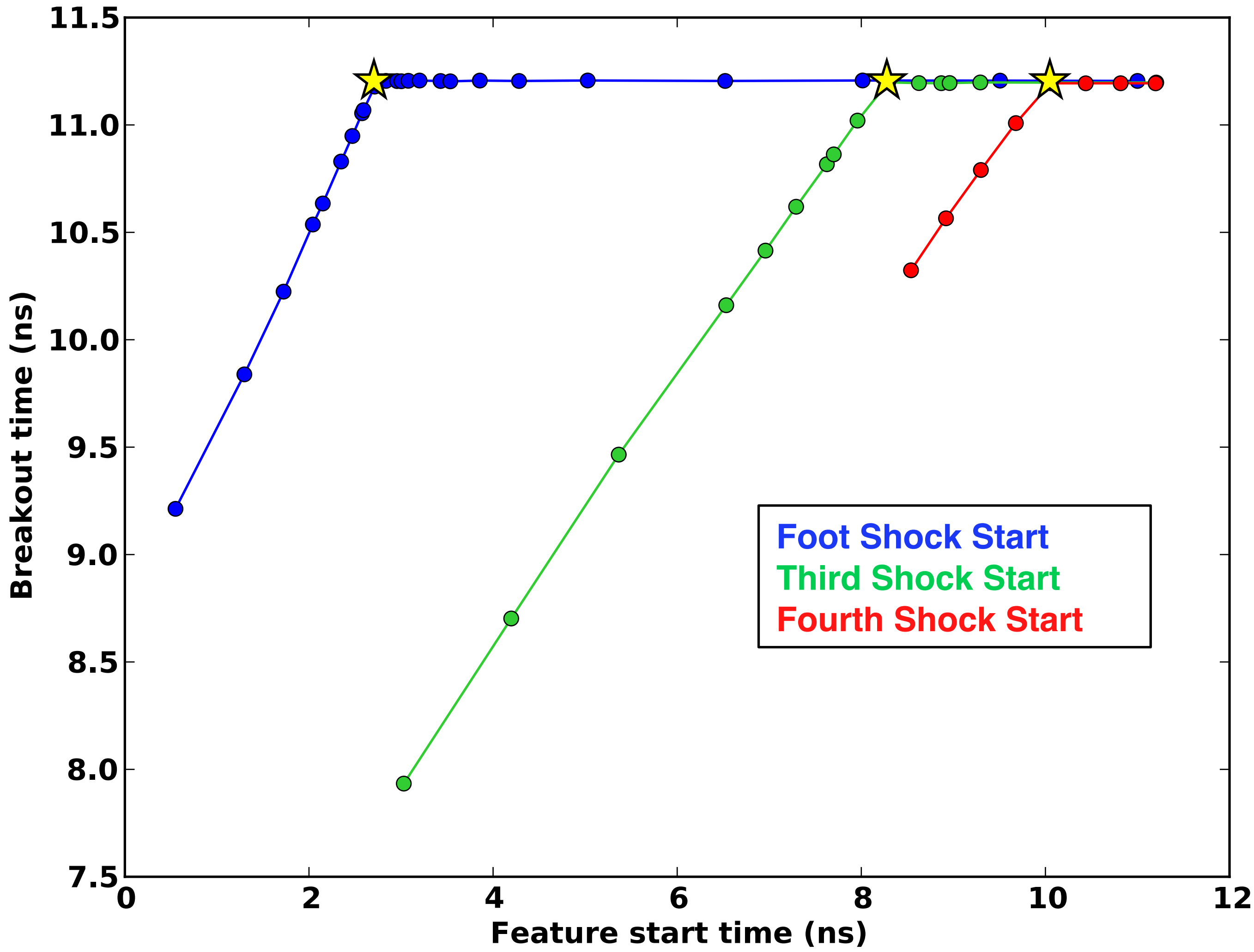
$$f_2(t) = t$$

$$f(t) = \omega t - b(t)$$

- **Experimentally tune ω to ensure good convergence, but introducing minimal error**
- **ω = 10% slope between first & last point works well**



Tuning method demonstrates breakout time unchanged after iteratively tuning of segments of pre-pulse

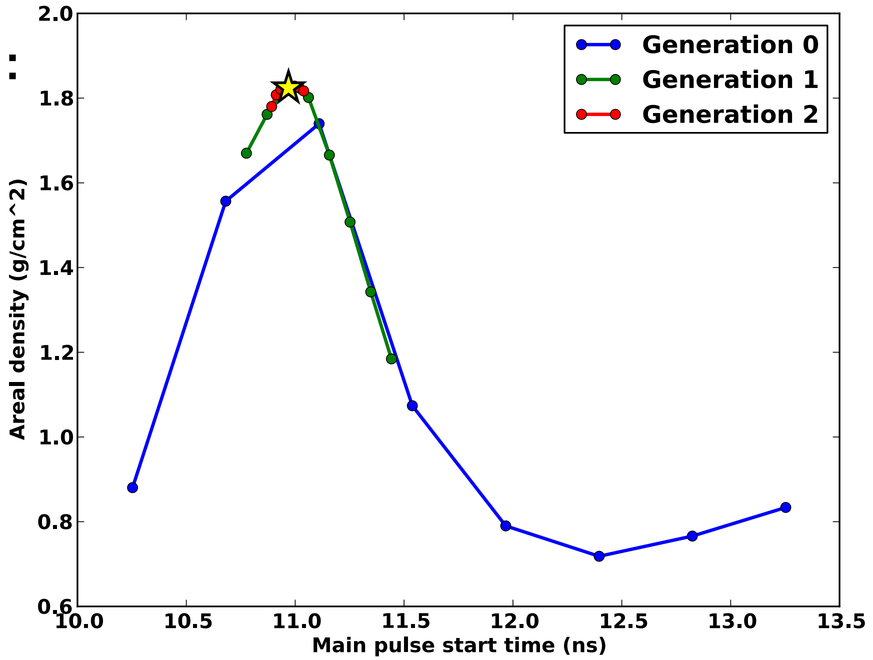


Main pulse is tuned to maximize burn fraction by maximizing areal density

- Having assembled the fuel to high density with the pre-pulse, the main pulse accelerates the shell to high velocity
- The duration of main pulse constrained by energy budget, so the onset of the main pulse must be timed to get the most “effective” acceleration
- Largest potential burn fraction is a good definitions of “effective”
- Assuming ignition, the fuel burn fraction scales with areal density (ρR) :

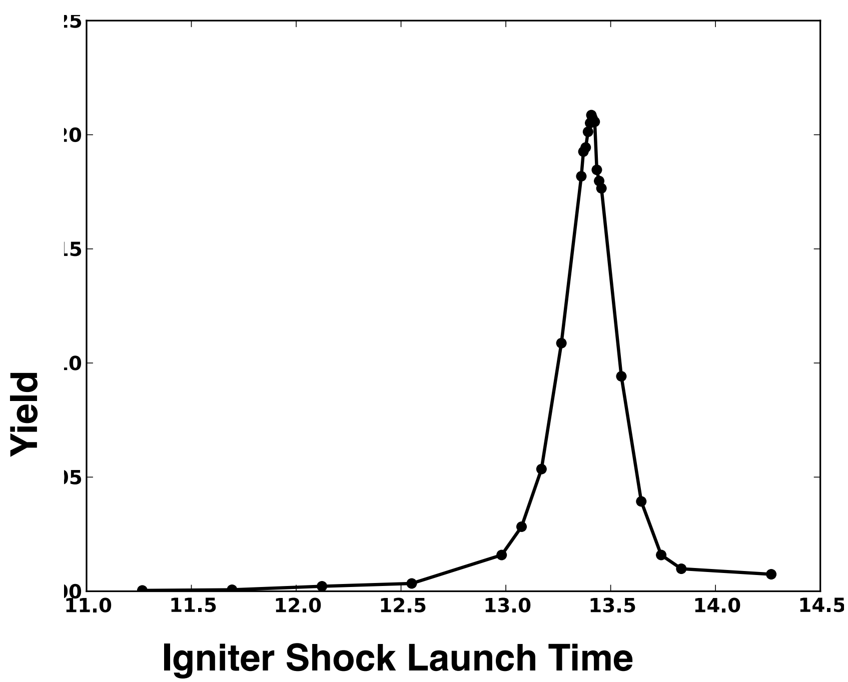
$$f \approx \frac{\rho R}{\rho R + 7}$$

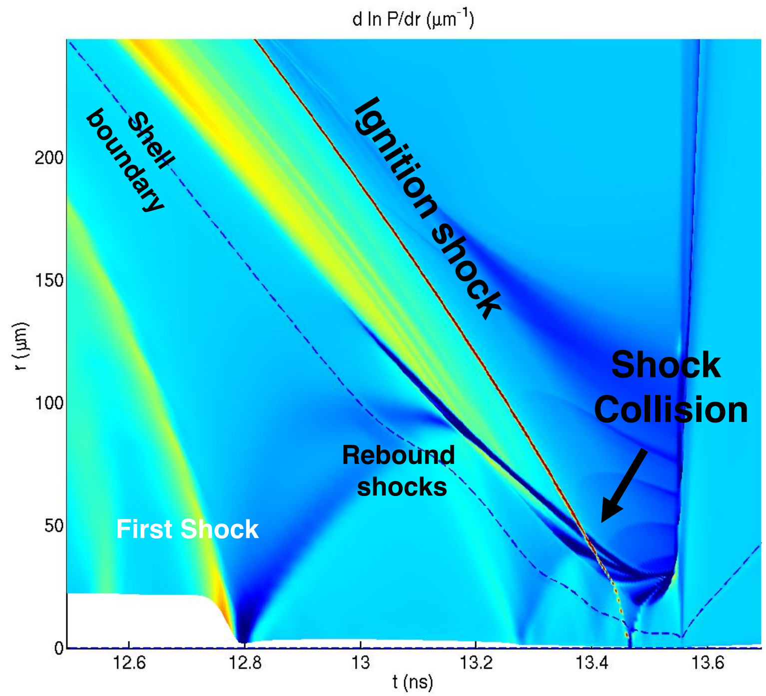
- Tune main pulse to maximize areal density



Igniter pulse is tuned to maximize thermonuclear yield

- Igniter shock should arrive near peak compression
- Too early and shock energy is diluted over large hot spot
- Too late and hot spot will disassemble before arrival of igniter shock
- Robust targets should have an "ignition window" where target burns robustly despite some mistiming





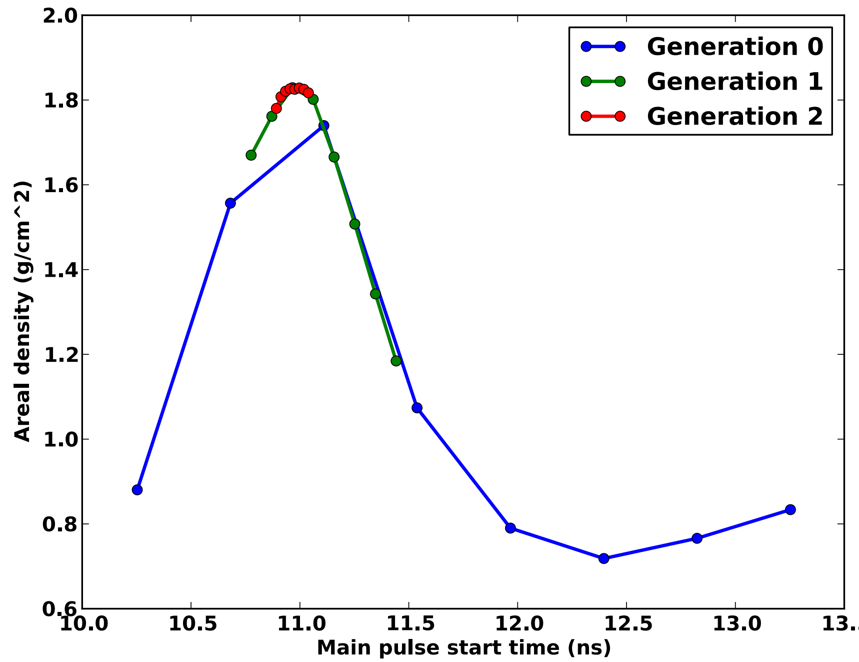
G. Schurtz X.Ribeyre et al.

Summary

- Manually tuning a laser pulse to a specific target is normally a labor intensive, high latency process
- Tuning can be automated given a sufficiently unambiguous description of "tuned"
- For laser shock ignition targets, the initial picket pulse sets the shock breakout time and other pre-pulse shocks should break out as close to this shock without changing the overall breakout time
- Main pulse should maximize areal density as to maximize burn fraction when ignited
- Igniter shock should be timed to maximize TN yield
- Timing algorithm assumes interaction between different pulse features is primarily hydrodynamic, allowing for tuning method to ignore interaction between features
- Future work
  - Autotune pedestal powers
  - Restart from prior completed calculations
  - Unfold experimental breakouts to infer pulse shape (inverse tuning)

Optimizations use a parallel 8-wide direct search method

- Function evaluations are relatively expensive (10-20 minutes) so want parallel method that minimizes number of iterations rather than number of function evaluations
- 8 parallel function evaluations per iteration (1 per processor)
- Start times equally spaced between end points
- If $f(x_i)$ is a maximum, refine end points to x_{i-1} and x_{i+1}



- Jobs within an iteration have nearly the same run-time, so processor utilization is high
- Converging timing error to < 50 ps typically takes 3-4 generations
  - ~1 hour wall time
- Method is simple to implement and has acceptable convergence rate