

---

# Corrfunc: Blazing fast correlation functions with SIMD Intrinsics

---

Dr. Manodeep Sinha

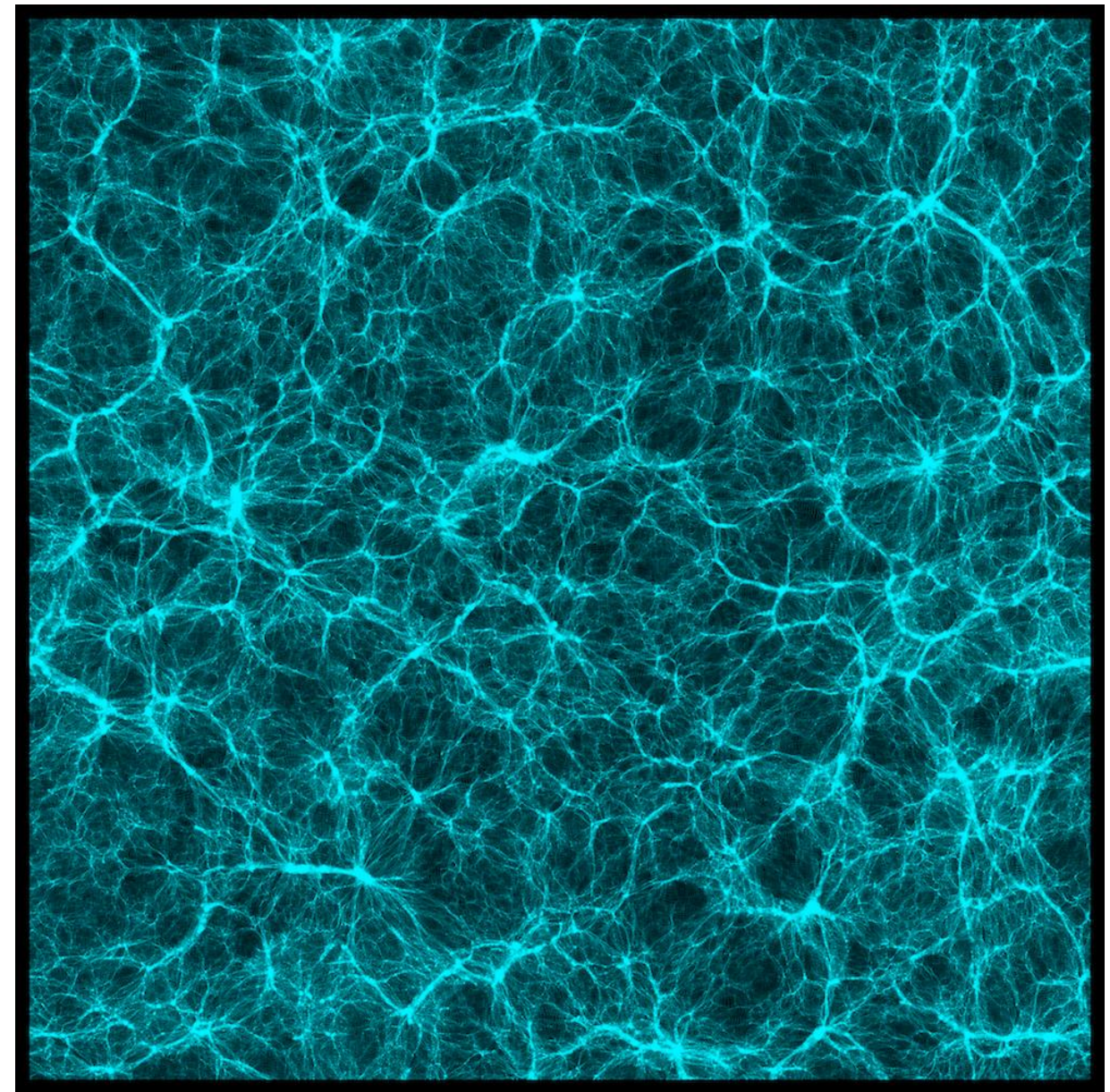
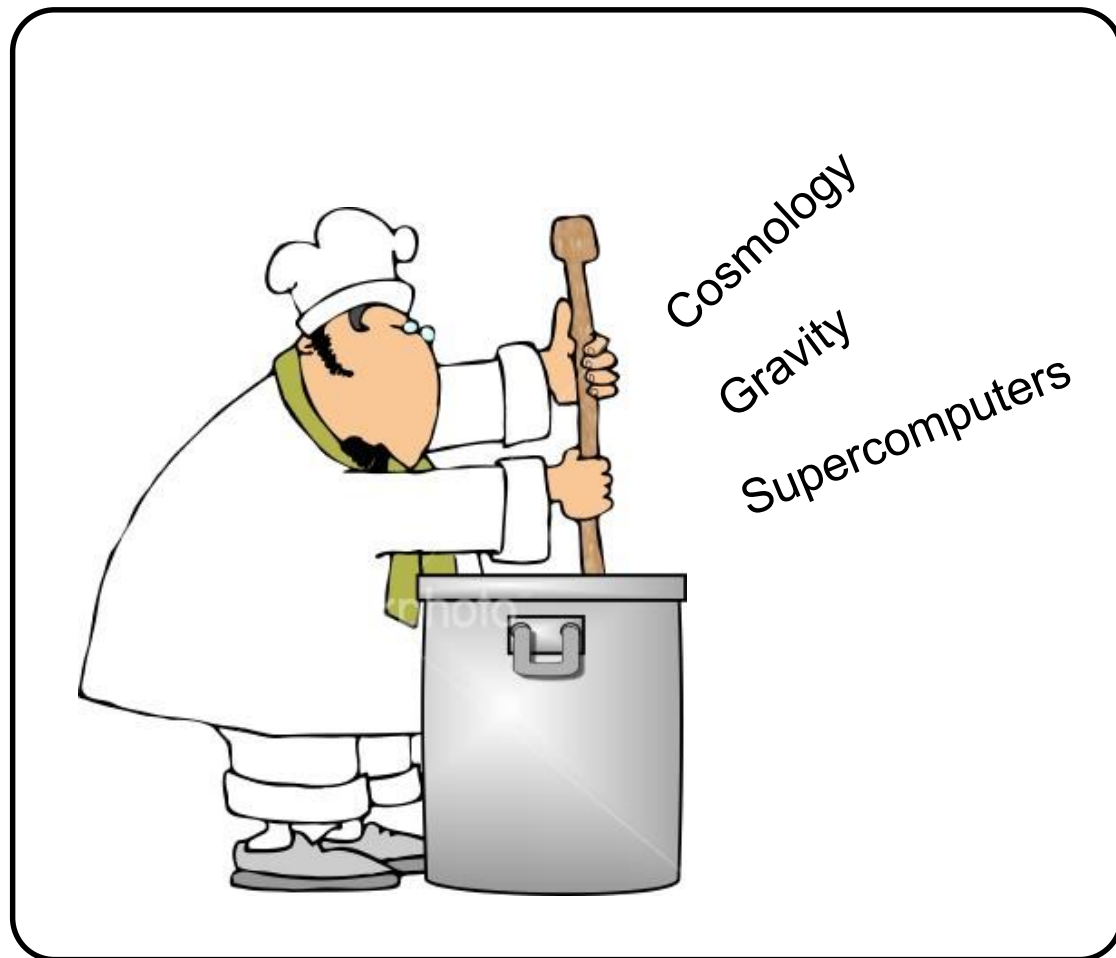
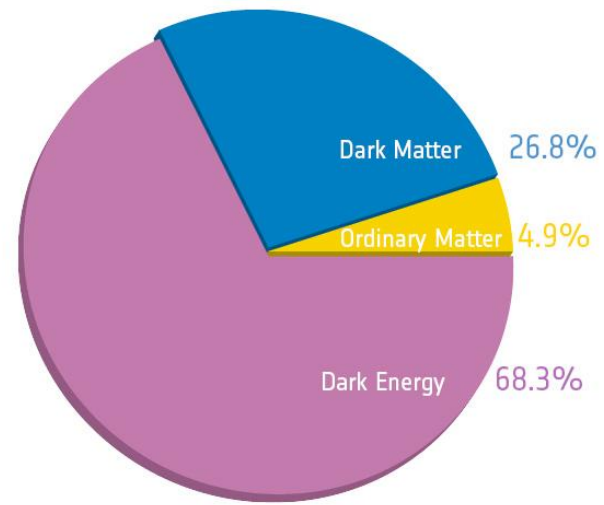
ASTRO 3D Centre of Excellence, Swinburne

Repo: [github.com/manodeep/Corrfunc/](https://github.com/manodeep/Corrfunc/)

Collaborators: Lehman Garrison

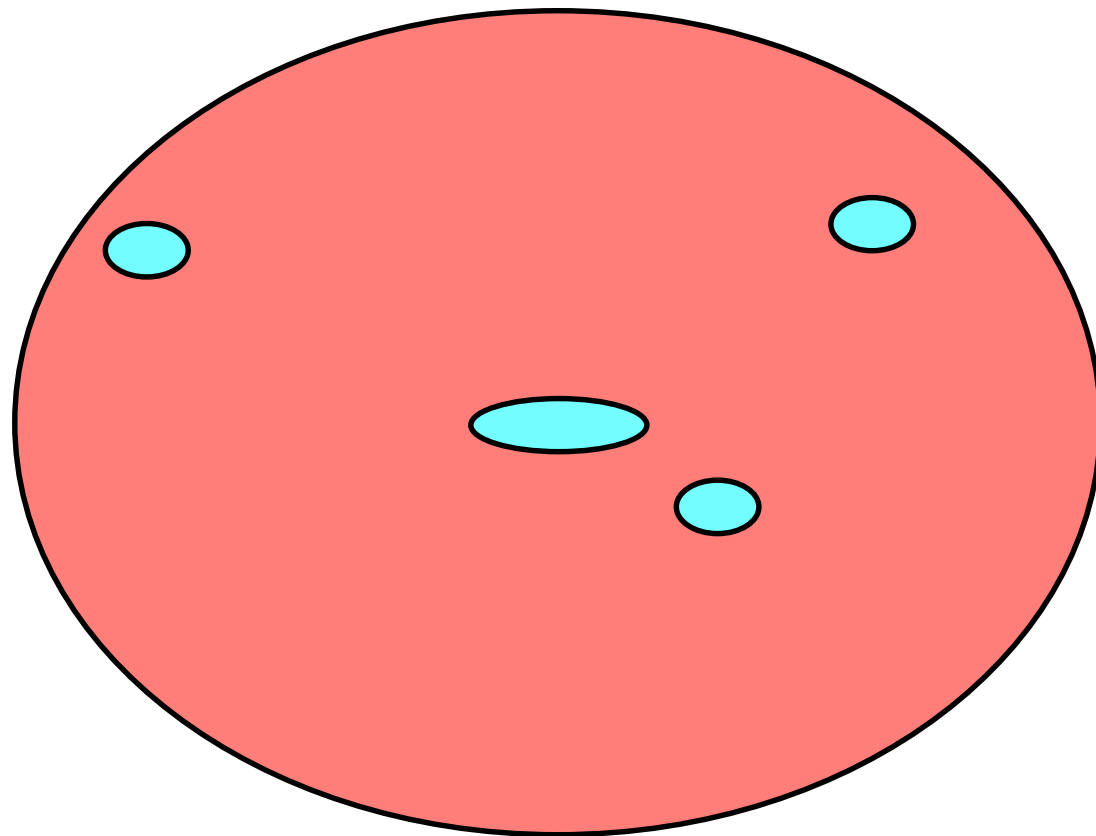
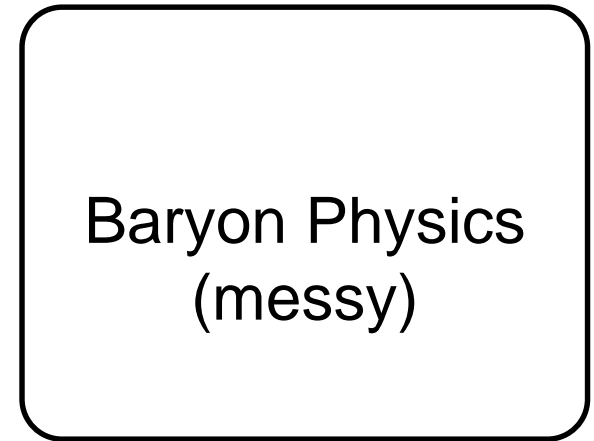
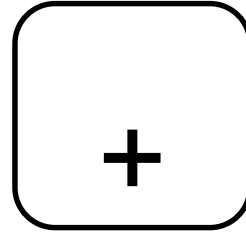
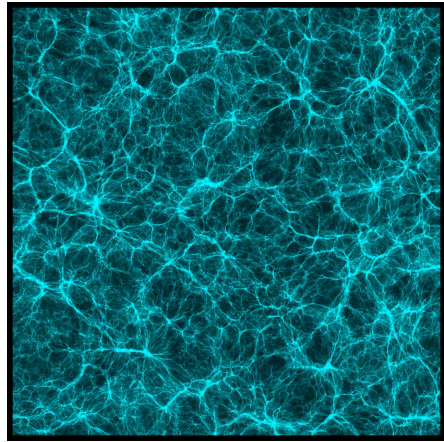
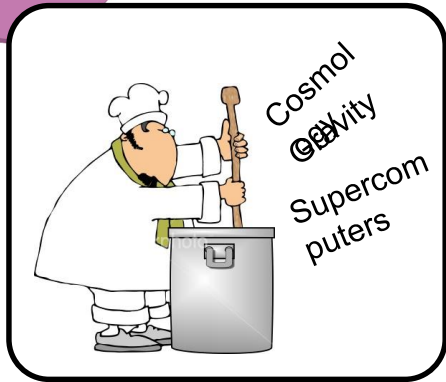
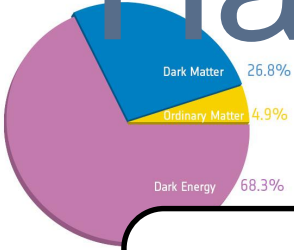
Contributors: Andrew Hearin, Nick Hand

# $\Lambda$ CDM Picture: Galaxies live in Halos



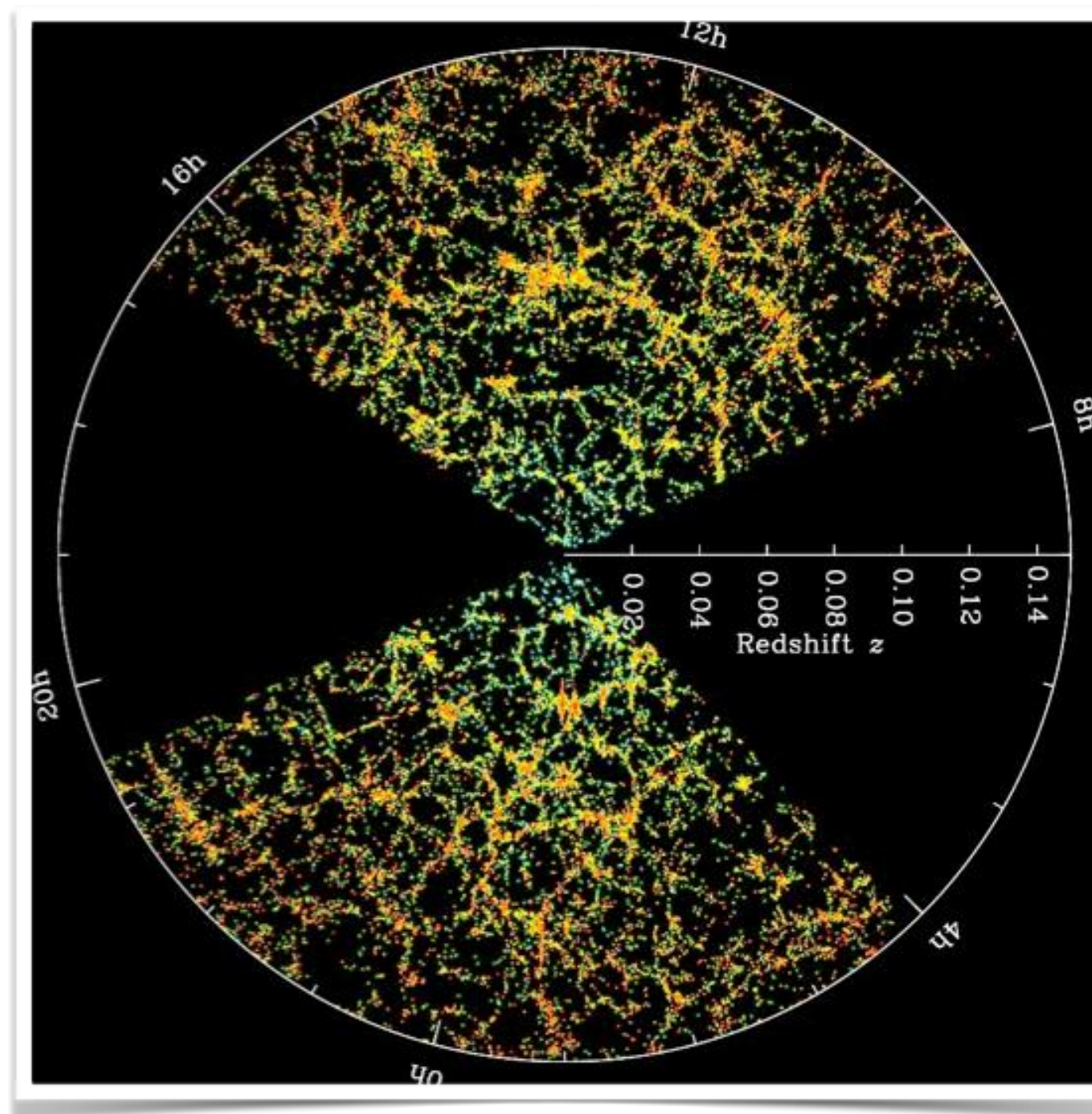
LasDamas Simulations, XSEDE/TACC

# $\Lambda$ CDM Picture: Galaxies live in Halos





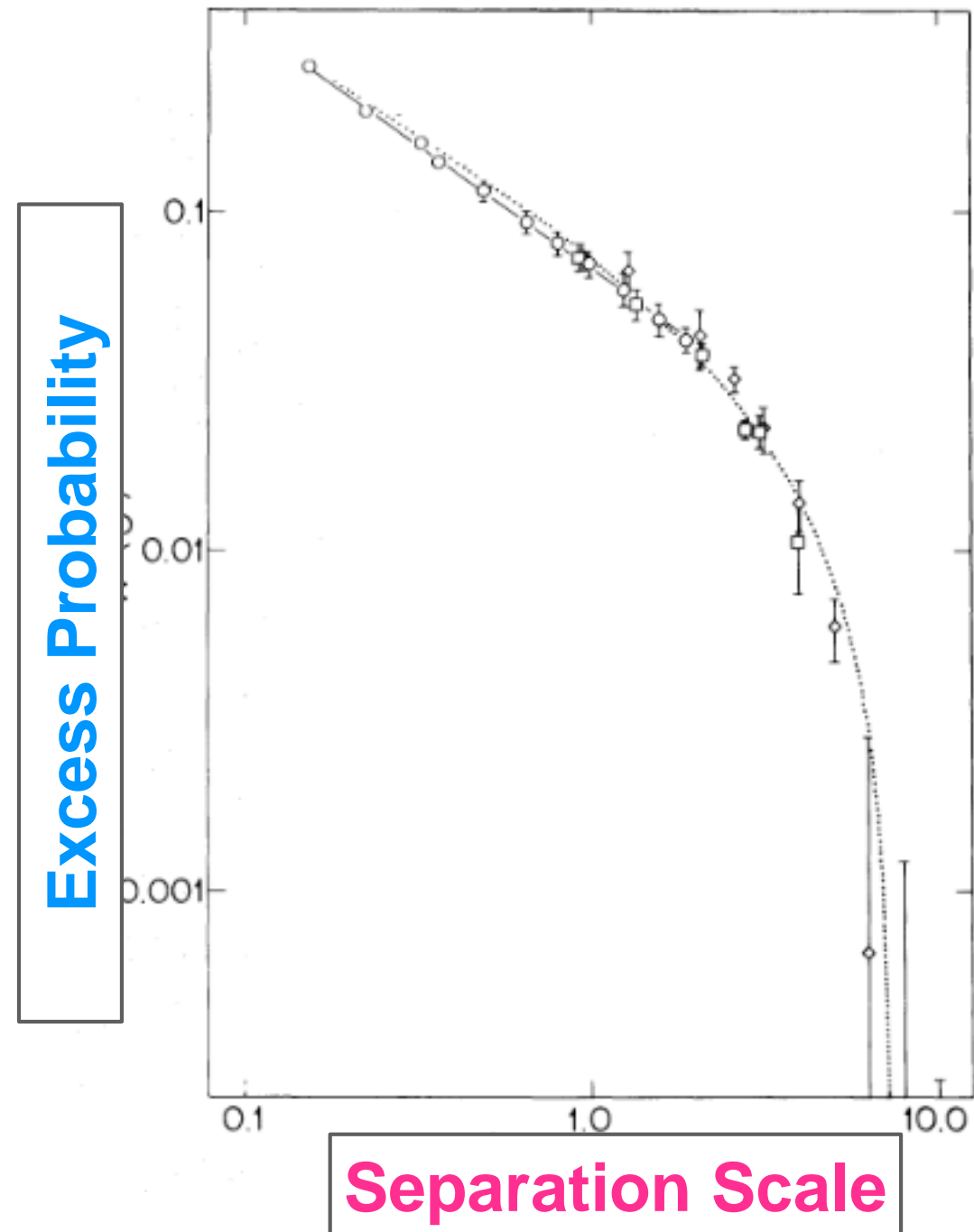
# Quantifying the Galaxy Distribution



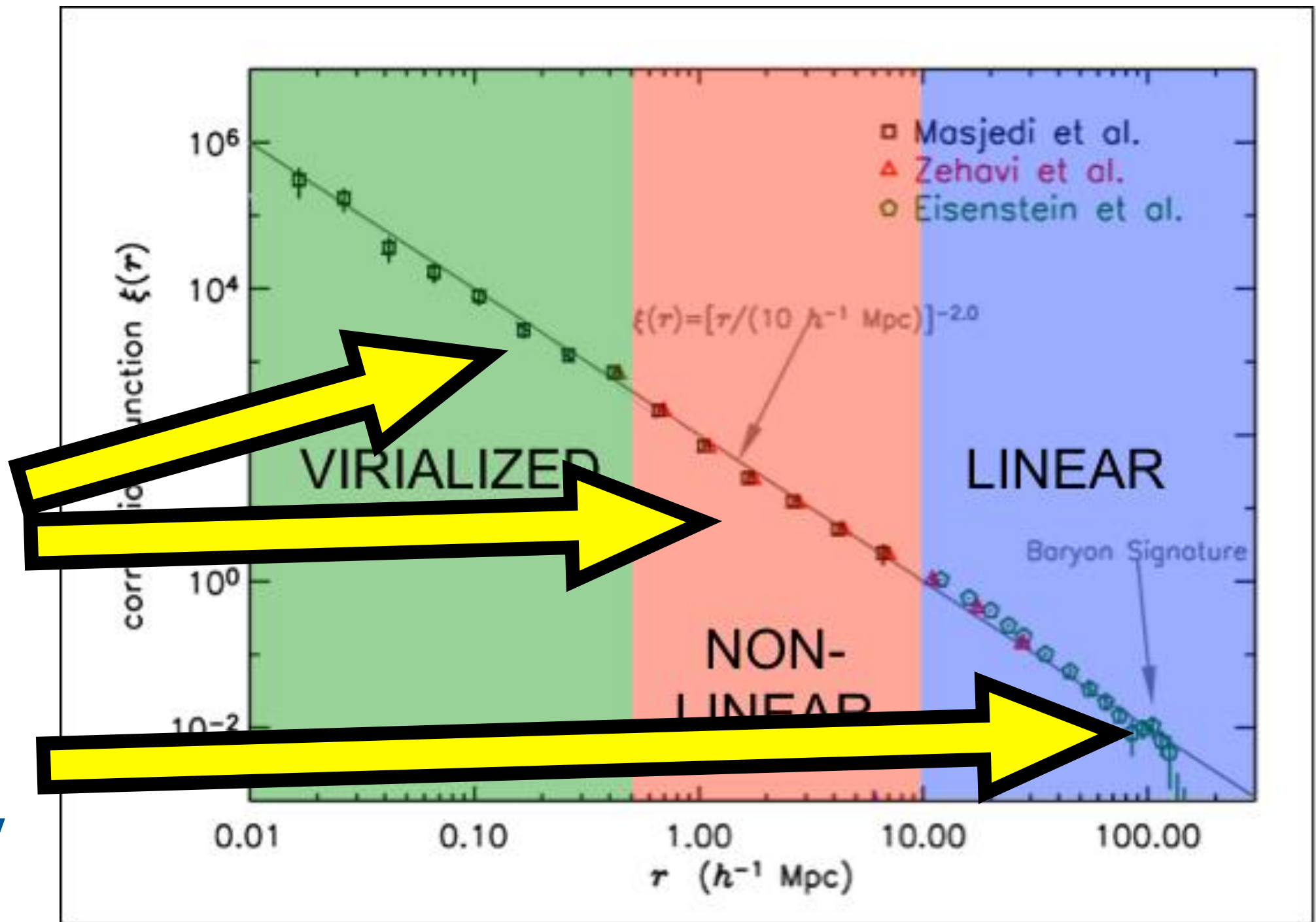
# What is a Correlation Function?

*Groth & Peebles, 1977*

- Measures the **excess probability** of finding a pair **at some separation**.



# Galaxy Clustering on different scales



Galaxy formation physics

Gravity + Cosmology

Masjedi et al. (2006)

Correlation functions are  
fundamental to understand how  
galaxies populate halos

# Code for a Correlation Function

```
for(int i=0;i<N1;i++) {  
  for(int j=0;j<N2;j++) {  
    double dist = @distance_metric@(point[i], point[j]);  
    if(dist < mindist || dist >= maxdist) {  
      continue;  
    }  
  
    int ibin = @dist_to_bin_index@(dist);  
    numpairs[ibin]++;  
  }  
}
```



# Code for a Correlation Function

```
for(int i=0;i<N1;i++) {  
  for(int j=0;j<N2;j++) {  
    double dist = @distance_metric@(point[i], point[j]);  
    if(dist < mindist || dist >= maxdist) {  
      continue;  
    }  
  
    int ibin = @dist_to_bin_index@(dist);  
    numpairs[ibin]++;  
  }  
}
```

# Simple Code is ... simple

- Ignores domain knowledge

(maxdist  $\ll$  L)

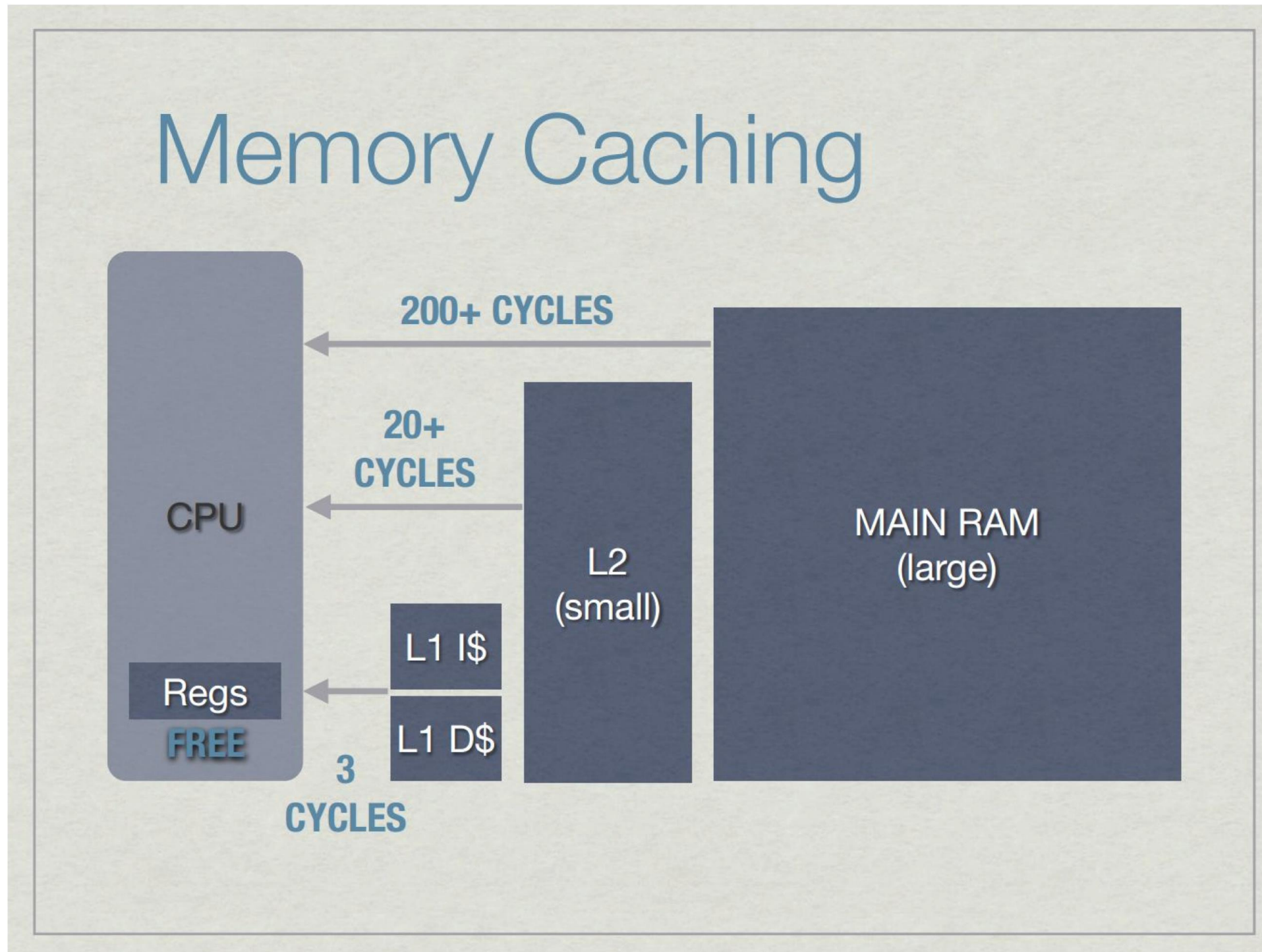
- Not optimal for hardware\*

- Can not be vectorized by  
compiler

```
for(int i=0;i<N1;i++) {  
  for(int j=0;j<N2;j++) {  
    double dist = @distance_metric@(point[i], point[j]);  
    if(dist < mindist || dist >= maxdist) {  
      continue;  
    }  
  
    int ibin = @dist_to_bin_index@(dist);  
    numpairs[ibin]++;  
  }  
}
```

# Hardware Detour

# Memory access is slow





# Memory access is slow

**How fast code runs  
depends on  
memory access  
patterns**



# Vector Instructions (SIMD)

## Scalar mode

(one instruction produces one result)

$a[i]$

+

$b[i]$

---

$a[i]+b[i]$

a

+

b

---

a+b

## SIMD processing

(one instruction can produce multiple results)

$a[i+7]$   $a[i+6]$   $a[i+5]$   $a[i+4]$   $a[i+3]$   $a[i+2]$   $a[i+1]$   $a[i]$

+

$b[i+7]$   $b[i+6]$   $b[i+5]$   $b[i+4]$   $b[i+3]$   $b[i+2]$   $b[i+1]$   $b[i]$

---

$c[i+7]$   $c[i+6]$   $c[i+5]$   $c[i+4]$   $c[i+3]$   $c[i+2]$   $c[i+1]$   $c[i]$

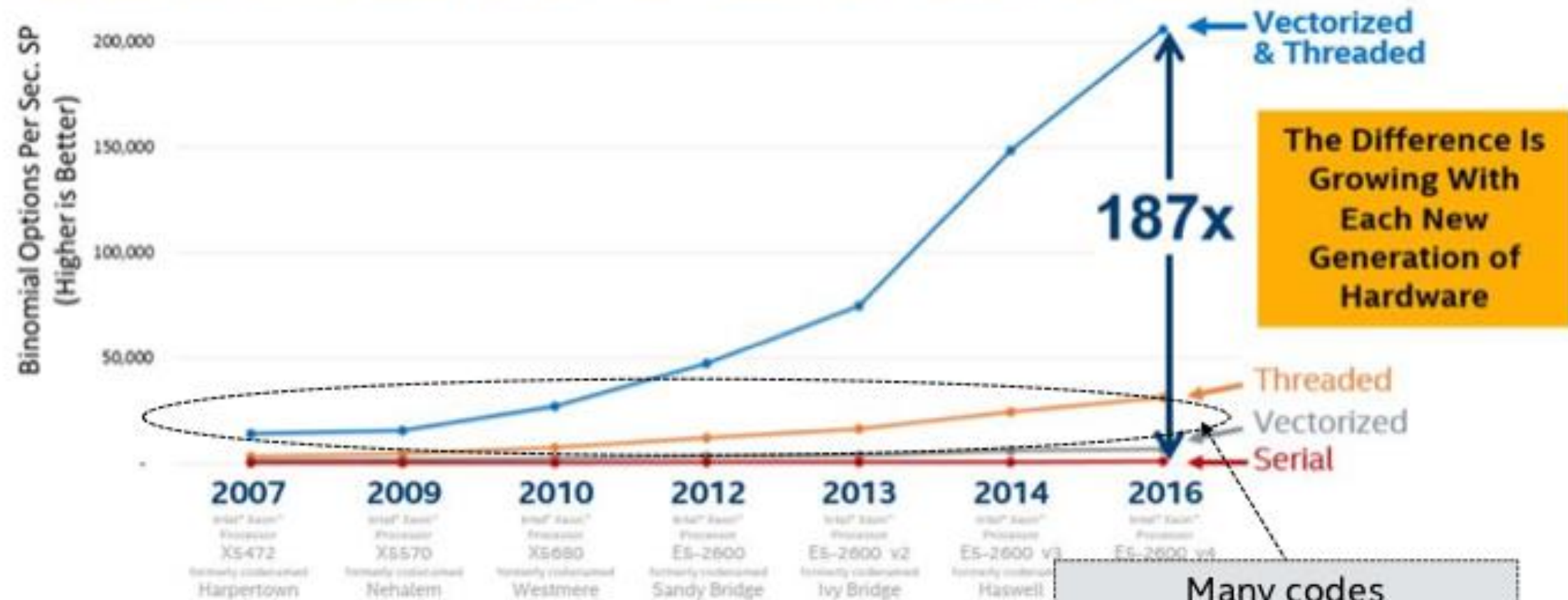
# Untapped Potential Can Be Huge!

[Configurations for Binomial Options SP](#)  
at the end  
of this presentation

## Vectorize & Thread or Performance Dies

Threaded + Vectorized can be much faster than either one alone

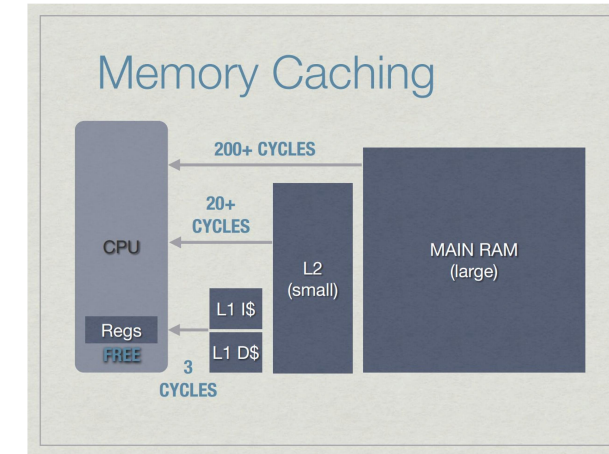
K. O'Leary, Intel



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of these factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to <http://www.intel.com/performance> at the end of this presentation

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of these factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to <http://www.intel.com/performance>

# Hardware → Performance



- Power scales as  $\text{freq}^3$ 
  - **Multi-cores at lower clock** (instead of one core with a 3GHz clock, 2 cores with 2.1 GHz provides 1.4x op/s @ 70% power)
- Memory access is slow
  - **Layers of (smaller, faster, dedicated) -> (larger, slower, shared) caches**
- Only one instruction per clock cycle
  - but, clock speeds have stalled
    - **More calculations per clock tick (SIMD/vectorization)**

**Vectorized operations with efficient memory access  
within independent kernels**



# Back to Corrfunc

# Simple Code is ... simple

- Ignores domain knowledge

(maxdist  $\ll$  L)

- Not optimal for hardware\*

- Can not be vectorized by  
compiler

```
for(int i=0;i<N1;i++) {  
  for(int j=0;j<N2;j++) {  
    double dist = @distance_metric@(point[i], point[j]);  
    if(dist < mindist || dist >= maxdist) {  
      continue;  
    }  
  
    int ibin = @dist_to_bin_index@(dist);  
    numpairs[ibin]++;  
  }  
}
```

# How Corrfunc works

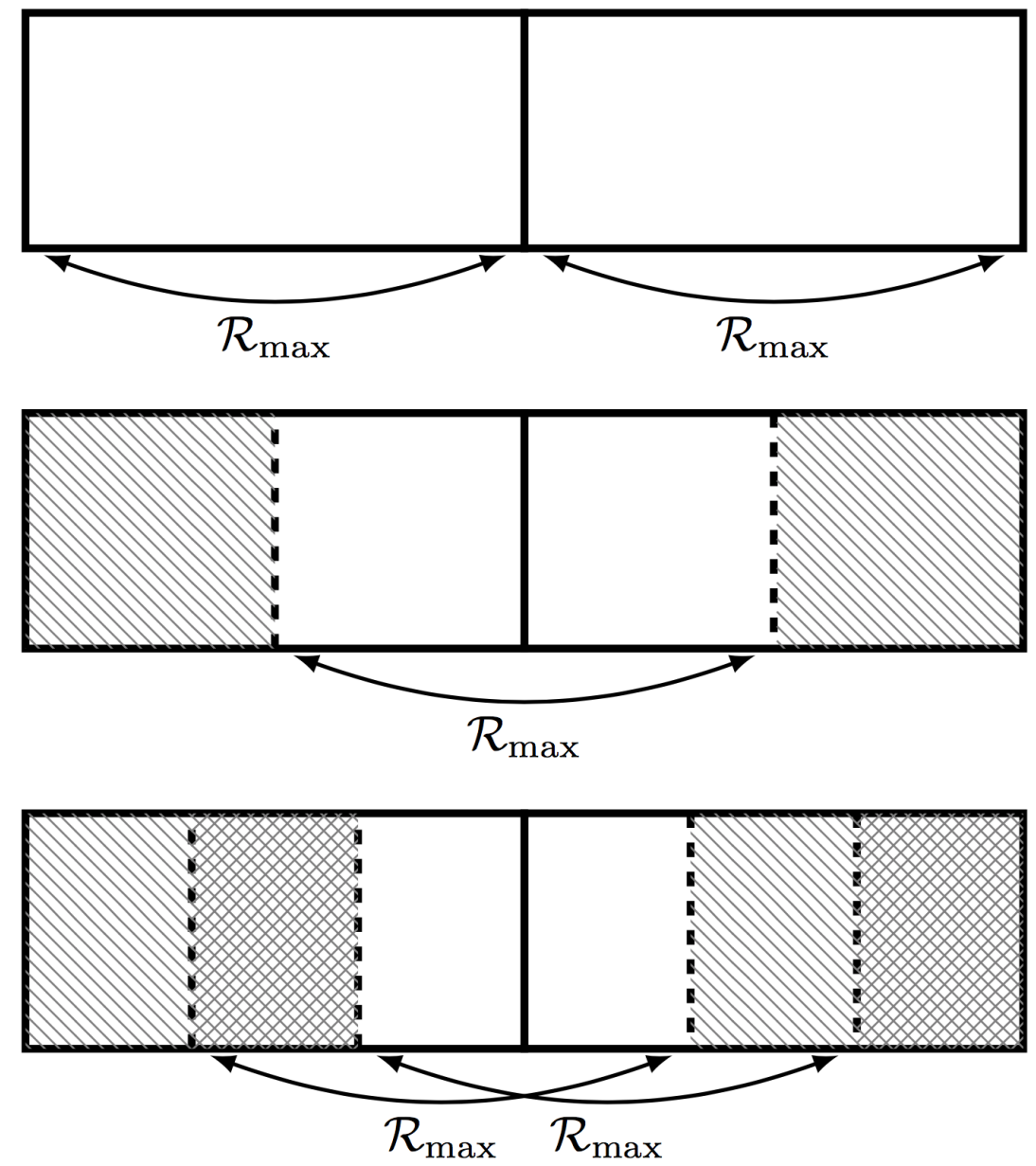
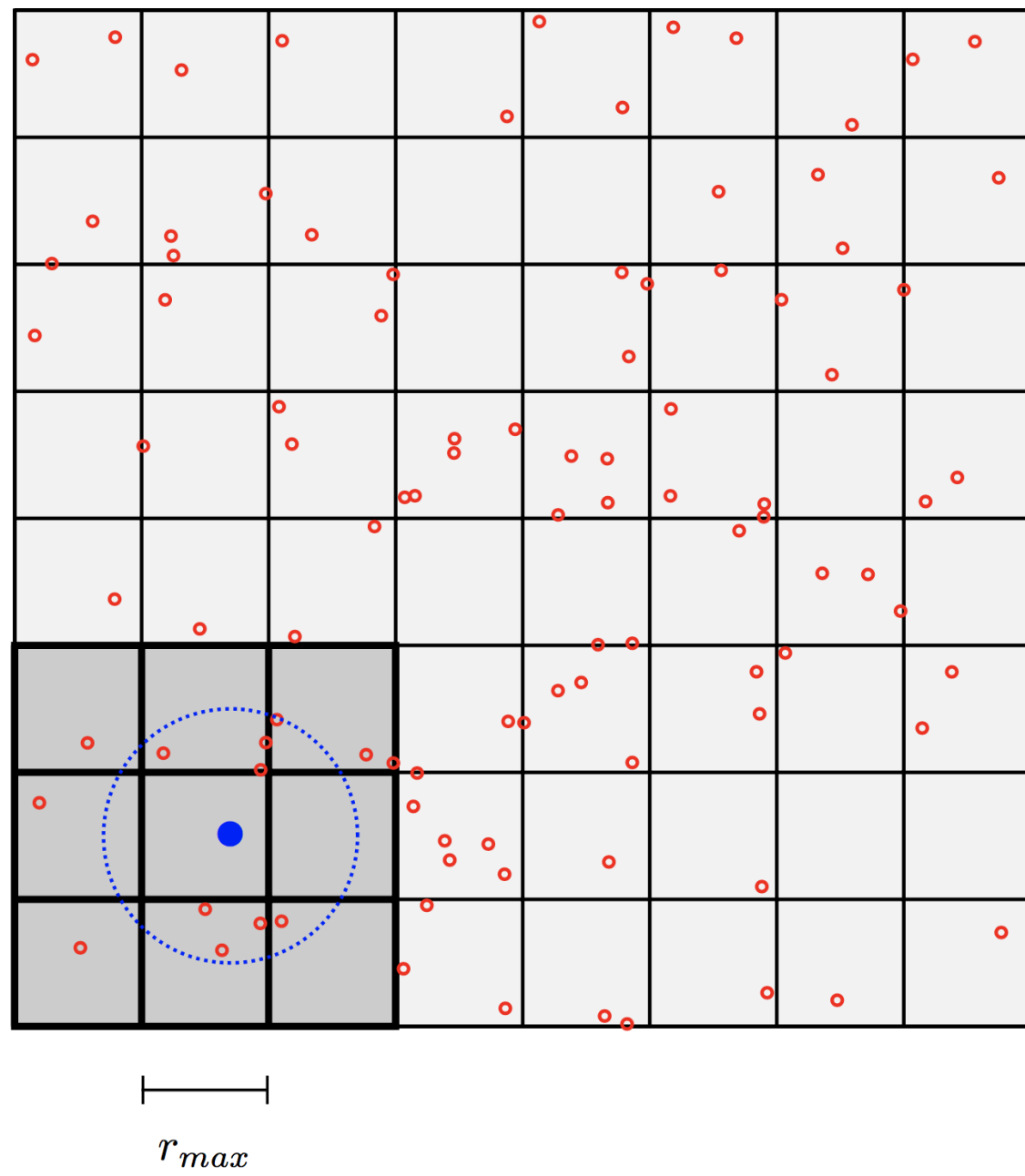
- Grids the particle distribution into 3D cells of size  $\sim R_{\max}$
- Stores particles contiguously within each cell
- Sorts particles within a cell in  $z$
- Only associates pairs of cells that **may** contain pairs
- Uses vectorised kernels on cell-pairs
- Outer OpenMP loop over cell-pairs

# Why Corrfunc is FAST

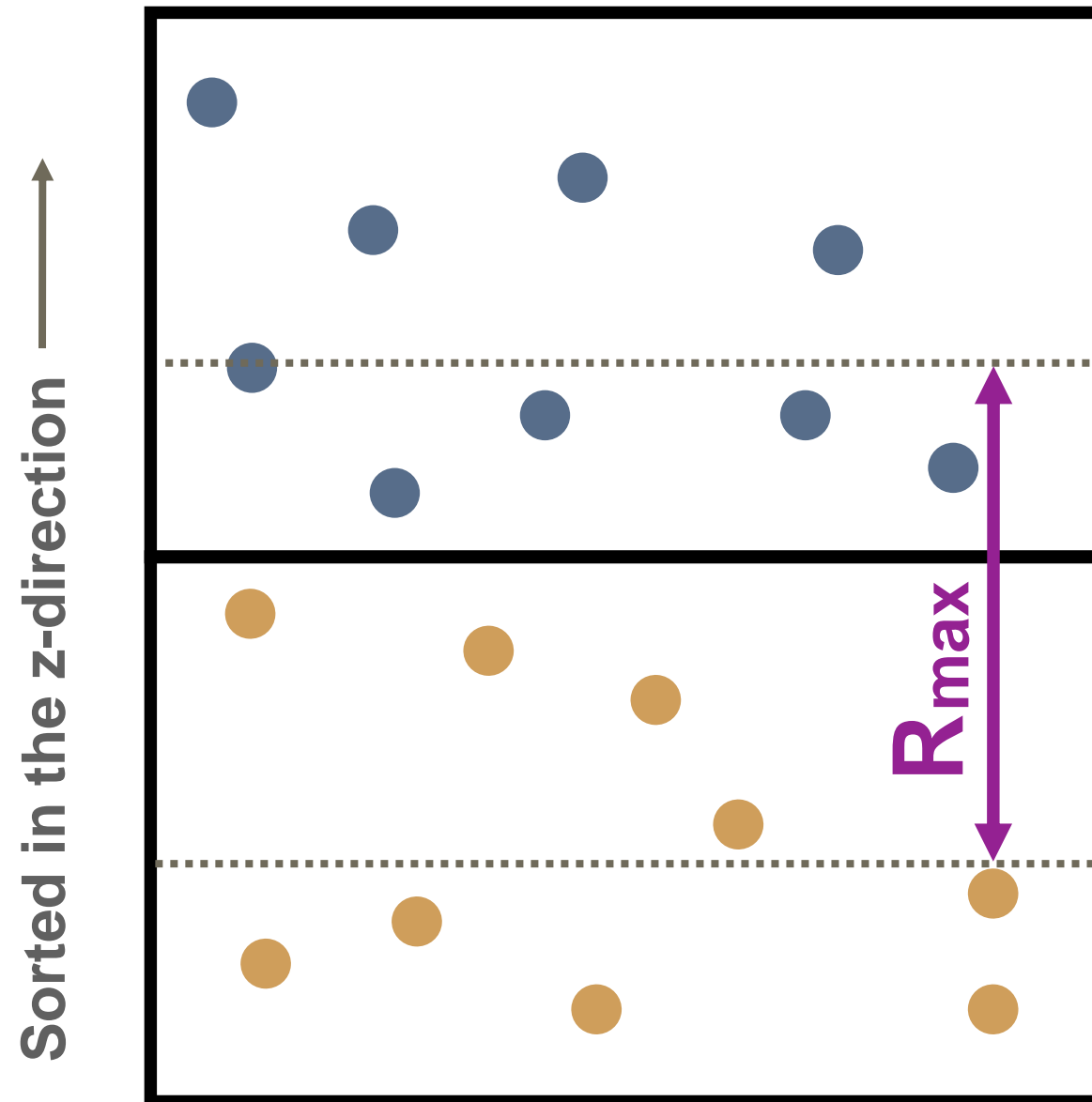
- Grids extent with cells of Rmax (**domain knowledge**)
- Stores particles contiguously within each cell (**memory access**)
- Uses sorting to prune (**algorithmic complexity**)
- Uses vector intrinsics (**vectorization**)
- Uses OpenMP (**multi-core**)



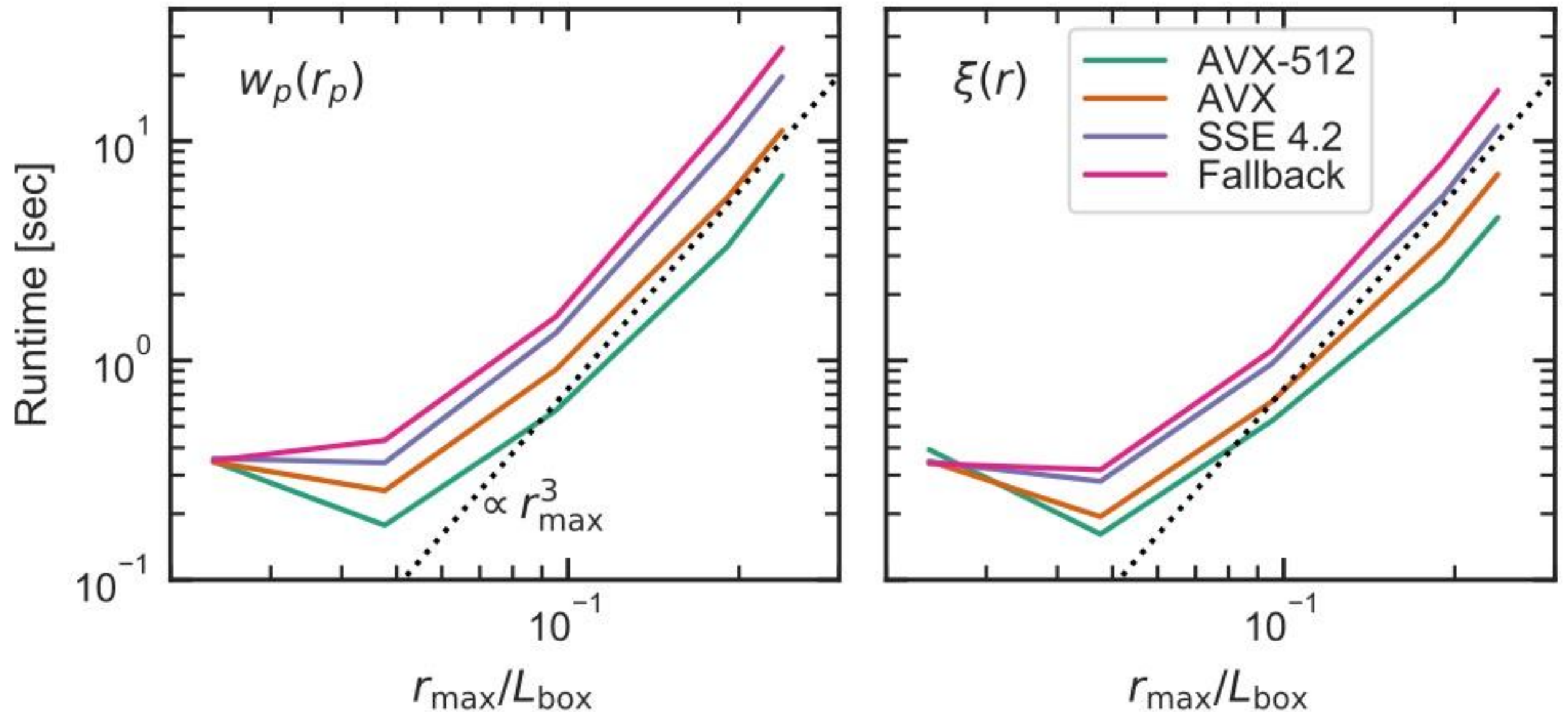
# Why Corrfunc is FAST: 3D Grid



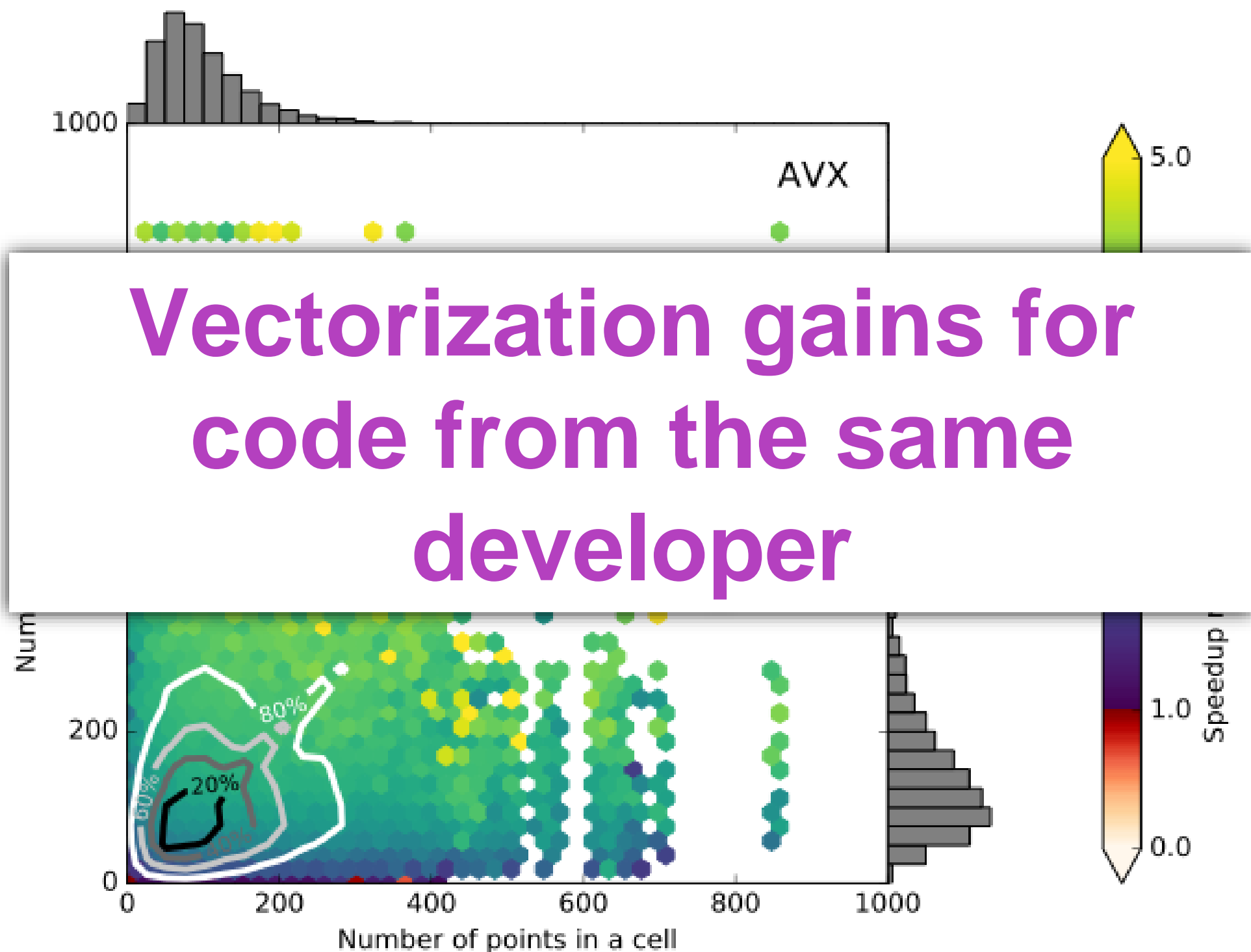
# Why Corrfunc is FAST: Sorting



# Performance of SIMD Kernels

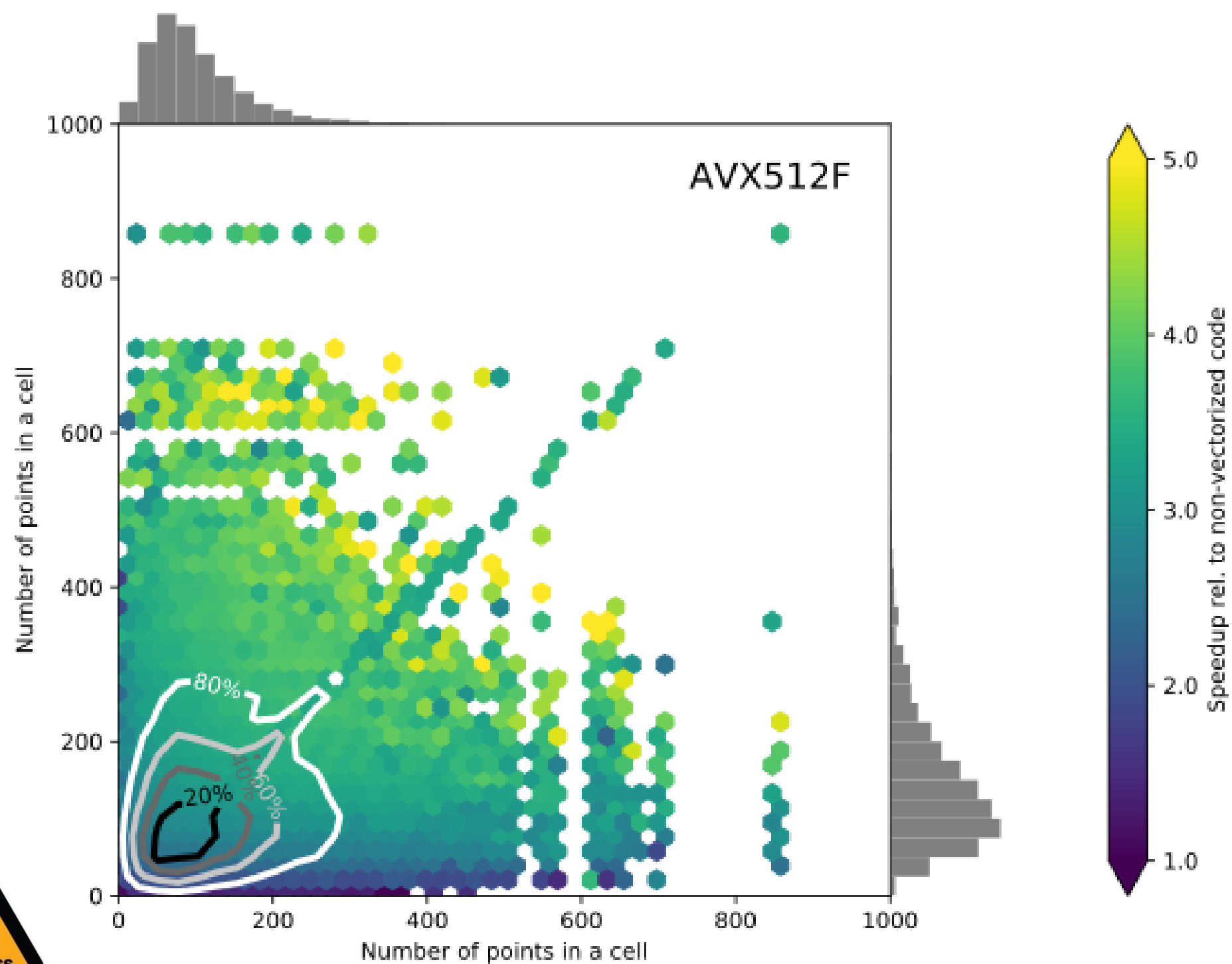


# Speedup from Vectorization (AVX)



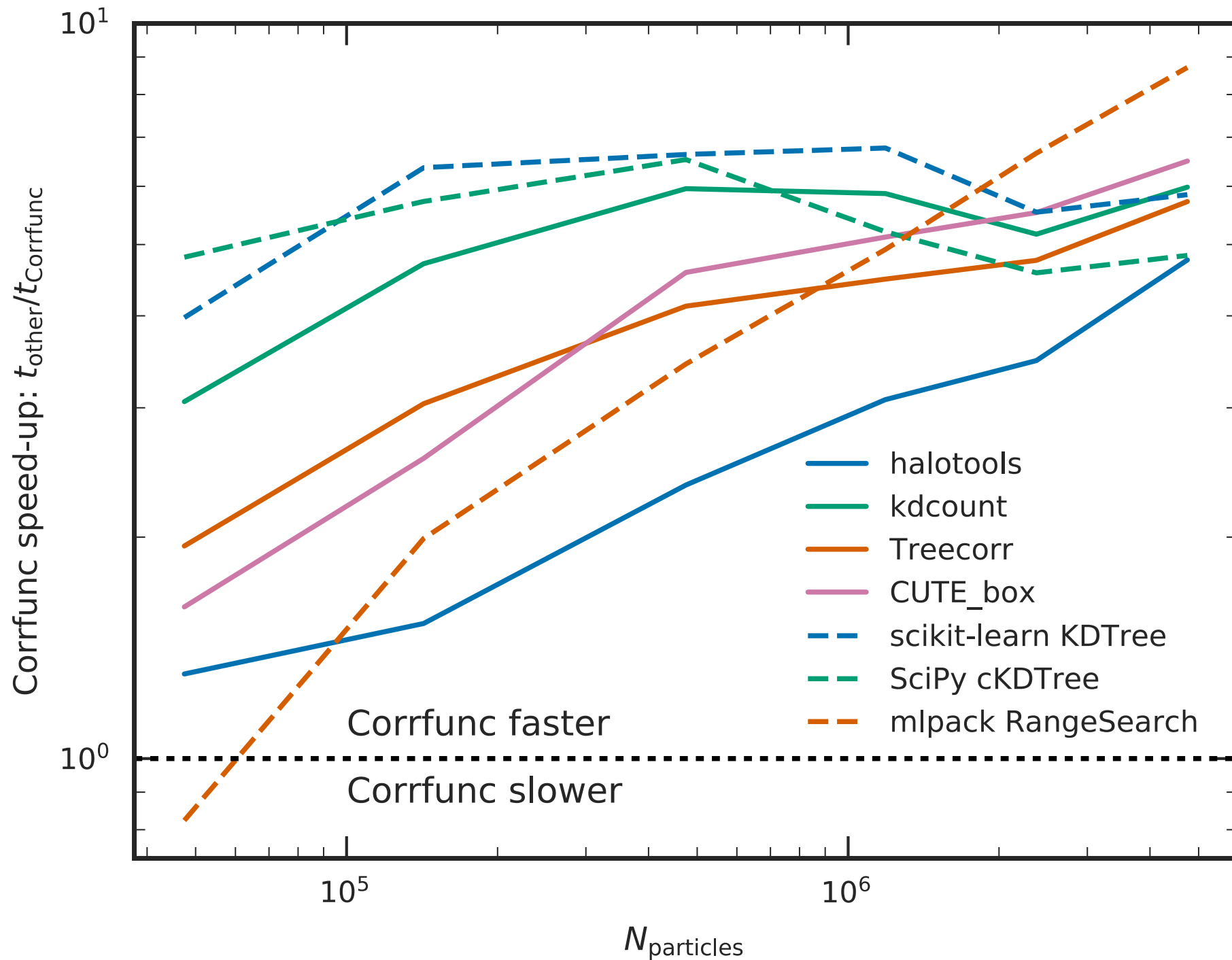


# Speedup from Vectorization (AVX512)



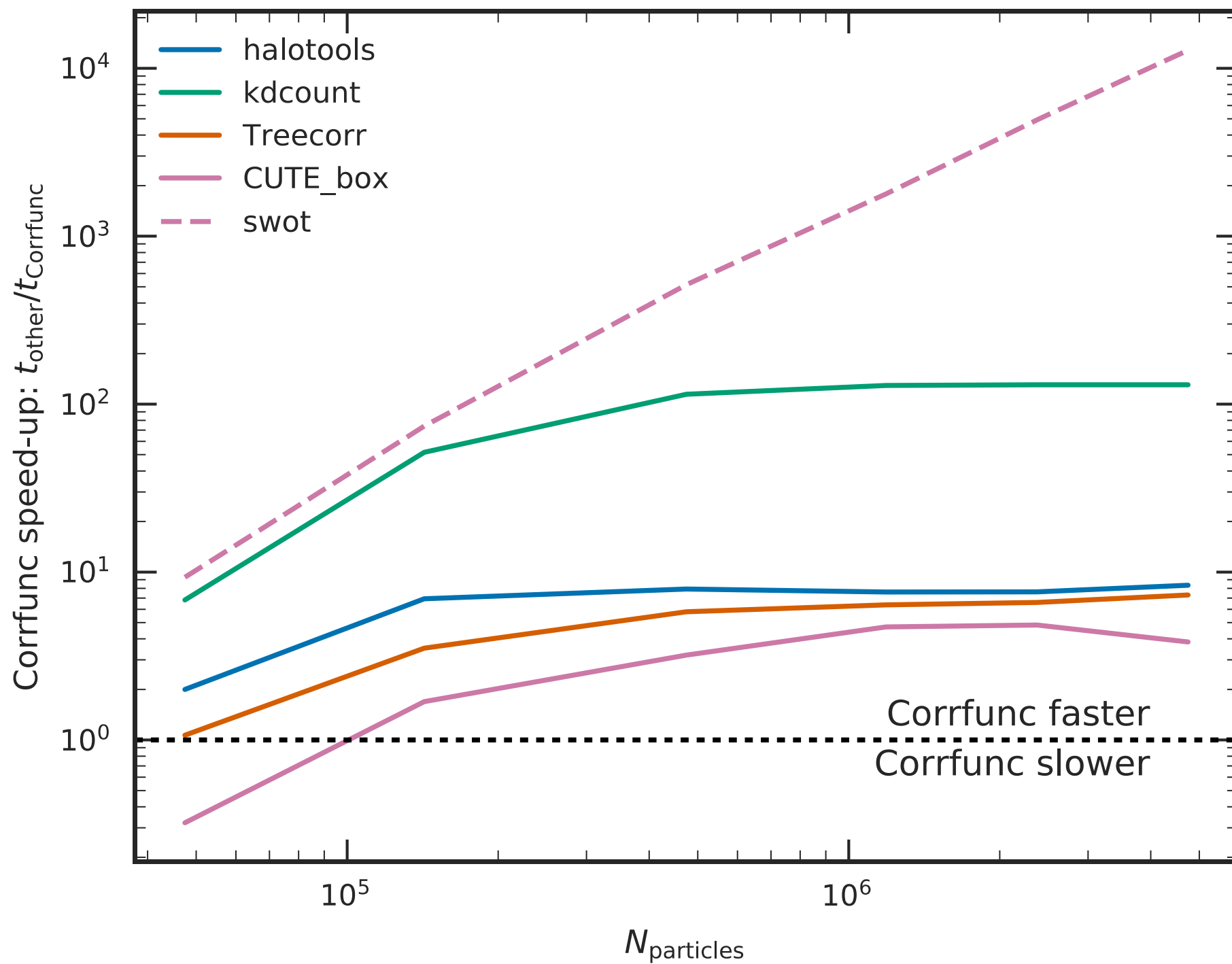
# Corrfunc Performance: Single-core

on github:  
[paper/scripts/generate\\_code\\_comparison.py](https://github.com/paper/scripts/generate_code_comparison.py)



# Corrfunc Performance: Multi-core

on github:  
[paper/scripts/generate\\_code\\_comparison.py](https://github.com/paper/scripts/generate_code_comparison.py)



# Why I wrote open-sourced Corrfunc

- Inherited codes took ~5 mins. MCMC would have exceeded the funding duration.
  - fast private version for my specific use-case
- Created custom code for experts with 6000x speedup (took < 24 hrs to create)
- Demonstrated the need for a fast, flexible, open-source package
- That initial 5 min calc. now takes ~5 secs with **Corrfunc**

# Writing Portable and Fast Software is Difficult


- Python removes the portability issue
  - but not fast
- Compiled extensions use very basic compiler options (defaults options are the ones used for compiling python)
- Compile with the highest compiler-supported ISA
  - Check ISA at runtime

**Usability/Sustainability trumps everything**

---

# Conclusions

---

- **Corrfunc** is optimised using domain knowledge, good memory access pattern, vectorization and OpenMP
- **Corrfunc** is “blazing fast” and
  - modular, user-friendly, documented, tested, OpenMP parallel, flexible API access, ...
  - GPU version coming - thanks to  ADACS
- my highest cited bib-entry for last year ([ascl.net/1703.003](https://ascl.net/1703.003))




- 
- [Find Similar Abstracts](#) (with [default settings below](#))
  - [Electronic On-line Article \(HTML\)](#)
  - [On-line Data](#)
  - [Citations to the Article \(17\)](#) ([Citation History](#))
  - [Refereed Citations to the Article](#)
  - [Associated Articles](#)
  - [Also-Read Articles](#) ([Reads History](#))
  - [Translate This Page](#)

**Title:** Corrfunc: Blazing fast correlation functions on the CPU  
**Authors:** [Sinha, Manodeep](#); [Garrison, Lehman](#)  
**Publication:** Astrophysics Source Code Library, record ascl:1703.003  
**Publication Date:** 03/2017  
**Origin:** ASCL  
**Keywords:** Software  
**Bibliographic Code:** [2017ascl.soft03003S](#)

---

# Conclusions

---

- **Corrfunc** is optimised using domain knowledge, good memory access pattern, vectorization and OpenMP
- **Corrfunc** is “blazing fast” and
  - modular, user-friendly, documented, tested, OpenMP parallel, flexible API access, ...
  - GPU version coming - thanks to  ADACS
- my highest cited bib-entry for last year ([ascl.net/1703.003](https://ascl.net/1703.003))

```

for(int64_t i=0;i<N0;i++) {
  const AVX512_FLOATS m_xpos = AVX512_SET_FLOAT(*x0++);
  const AVX512_FLOATS m_ypos = AVX512_SET_FLOAT(*y0++);
  const AVX512_FLOATS m_zpos = AVX512_SET_FLOAT(*z0++);
  DOUBLE *localx1 = x1, *localy1 = y1, *localz1 = z1;
  for(int64_t j=0;j<N1;j++) {
    AVX512_MASK m_mask_left = (N1 - j) >= AVX512_NVEC ? ~0:masks_per_misalignment_value_DOUBLE[N1-j];
    const AVX512_FLOATS m_x1 = AVX512_MASKZ_LOAD_FLOATS_UNALIGNED(m_mask_left, localx1);
    const AVX512_FLOATS m_y1 = AVX512_MASKZ_LOAD_FLOATS_UNALIGNED(m_mask_left, localy1);
    const AVX512_FLOATS m_z1 = AVX512_MASKZ_LOAD_FLOATS_UNALIGNED(m_mask_left, localz1);

    /* this might actually exceed the allocated range but we will never dereference that */
    localx1 += AVX512_NVEC;
    localy1 += AVX512_NVEC;
    localz1 += AVX512_NVEC;

    const AVX512_FLOATS m_xdiff = AVX512_SUBTRACT_FLOATS(m_x1, m_xpos); /* (x[j:j+NVEC-1] - x0) */
    const AVX512_FLOATS m_ydiff = AVX512_SUBTRACT_FLOATS(m_y1, m_ypos); /* (y[j:j+NVEC-1] - y0) */
    const AVX512_FLOATS m_zdiff = AVX512_SUBTRACT_FLOATS(m_z1, m_zpos); /* z[j:j+NVEC-1] - z1 */

    const AVX512_FLOATS m_sqr_xdiff = AVX512_SQUARE_FLOAT(m_xdiff); /* (x0 - x[j])^2 */
    const AVX512_FLOATS x2py2 = AVX512_FMA_ADD_FLOATS(m_ydiff, m_ydiff, m_sqr_xdiff); /* dy*dy + dx^2 */
    const AVX512_FLOATS r2 = AVX512_FMA_ADD_FLOATS(m_zdiff, m_zdiff, x2py2); /* dz*dz + (dy^2 + dx^2) */
    const AVX512_MASK m_rpmask_mask = AVX512_MASK_COMPARE_FLOATS(m_mask_pimax, r2, m_sqr_rpmask, _CMP_LT_OQ);
    /* Create a combined mask */
    /* This gives us the mask for all sqr_rpmask <= r2 < sqr_rpmask */
    m_mask_left = AVX512_MASK_COMPARE_FLOATS(m_rpmask_mask, r2, m_sqr_rpmask, _CMP_GE_OQ);
    if(m_mask_left == 0) {
      continue;
    }
    /* Loop backwards through nbins. m_mask_left contains all the points that */
    /* are less than rpmask at the beginning of the loop. */
    for(int kbin=nbins-1;kbin>=1;kbin--) {
      const AVX512_MASK m_bin_mask = AVX512_MASK_COMPARE_FLOATS(m_mask_left, r2,m_rupp_sqr[kbin-1],_CMP_GE_OS);
      npairs[kbin] += bits_set_in_avx512_mask_DOUBLE[m_bin_mask];
      /* ANDNOT(X, Y) -> NOT X AND Y */
      m_mask_left = AVX512_MASK_BITWISE_AND_NOT(m_bin_mask, m_mask_left);
      if(m_mask_left == 0) {
        break;
      }
    }
  }
}

```

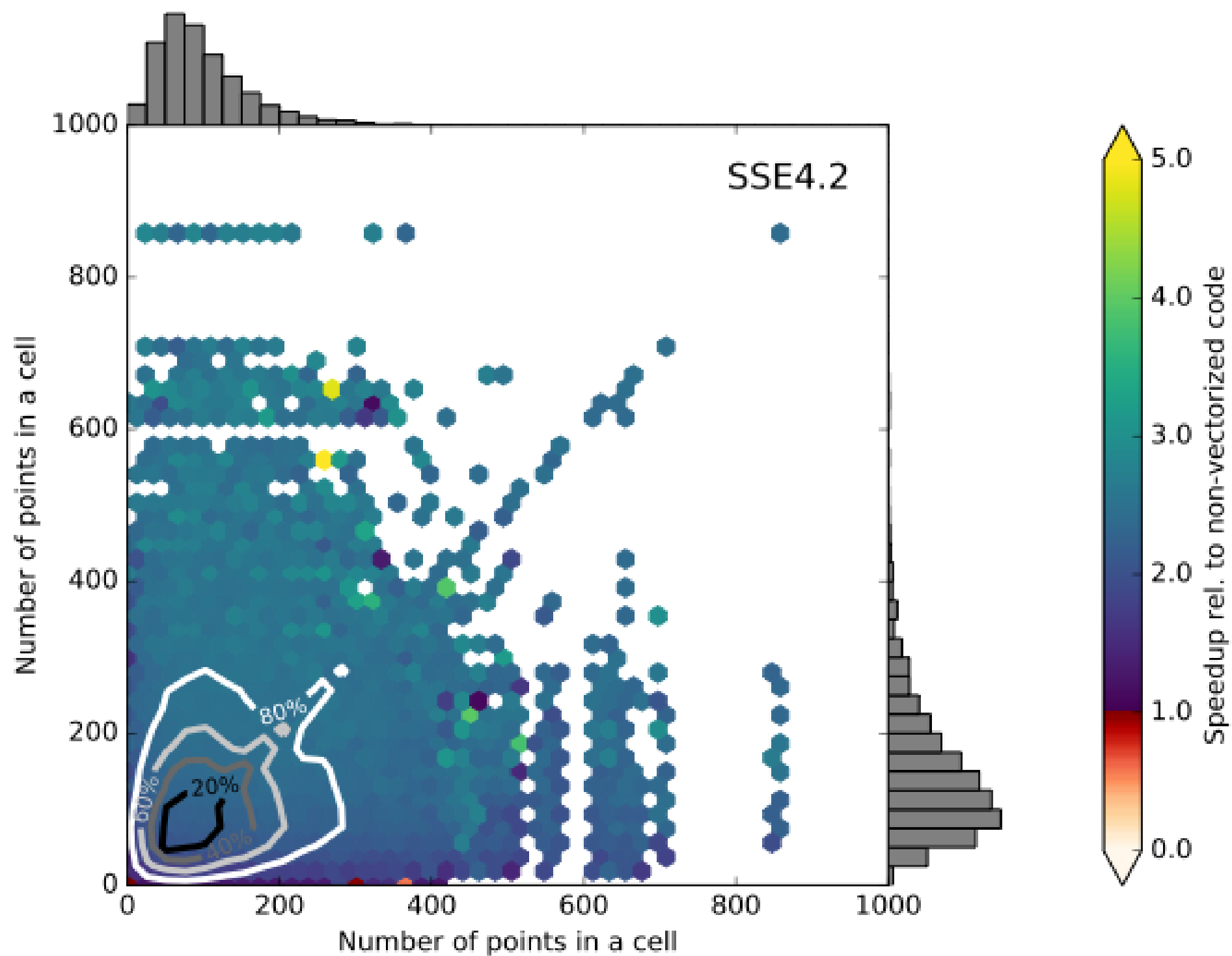
---

# Corrfunc Kernel

---

<https://gist.github.com/manodeep/ffdc60024fd6df8b5264657f0be2f967>

# Speedup from Vectorization (SSE4.2)



```

for(int64_t i=0;i<N0;i++) {
const AVX512_FLOATS m_xpos = AVX512_SET_FLOAT(*x0++);
const AVX512_FLOATS m_ypos = AVX512_SET_FLOAT(*y0++);
const AVX512_FLOATS m_zpos = AVX512_SET_FLOAT(*z0++);
DOUBLE *localx1 = x1, *localy1 = y1, *localz1 = z1;
for(int64_t j=0;j<N1;j++) {
AVX512_MASK m_mask_left = (N1 - j) >= AVX512_NVEC ? ~0:masks_per_misalignment_value_DOUBLE[N1-j];
const AVX512_FLOATS m_x1 = AVX512_MASKZ_LOAD_FLOATS_UNALIGNED(m_mask_left, localx1);
const AVX512_FLOATS m_y1 = AVX512_MASKZ_LOAD_FLOATS_UNALIGNED(m_mask_left, localy1);
const AVX512_FLOATS m_z1 = AVX512_MASKZ_LOAD_FLOATS_UNALIGNED(m_mask_left, localz1);

/* this might actually exceed the allocated range but we will never dereference that */
localx1 += AVX512_NVEC;
localy1 += AVX512_NVEC;
localz1 += AVX512_NVEC;

const AVX512_FLOATS m_xdiff = AVX512_SUBTRACT_FLOATS(m_x1, m_xpos); /* (x[j:j+NVEC-1] - x0) */
const AVX512_FLOATS m_ydiff = AVX512_SUBTRACT_FLOATS(m_y1, m_ypos); /* (y[j:j+NVEC-1] - y0) */
const AVX512_FLOATS m_zdiff = AVX512_SUBTRACT_FLOATS(m_z1, m_zpos); /* z[j:j+NVEC-1] - z1 */

const AVX512_FLOATS m_sqr_xdiff = AVX512_SQUARE_FLOAT(m_xdiff); /* (x0 - x[j])^2 */
const AVX512_FLOATS x2py2 = AVX512_FMA_ADD_FLOATS(m_ydiff, m_ydiff, m_sqr_xdiff); /* dy*dy + dx^2 */
const AVX512_FLOATS r2 = AVX512_FMA_ADD_FLOATS(m_zdiff, m_zdiff, x2py2); /* dz*dz + (dy^2 + dx^2) */
const AVX512_MASK m_rpmask_mask = AVX512_MASK_COMPARE_FLOATS(m_mask_pimax, r2, m_sqr_rpmask, _CMP_LT_OQ);
/* Create a combined mask */
/* This gives us the mask for all sqr_rpmask <= r2 < sqr_rpmask */
m_mask_left = AVX512_MASK_COMPARE_FLOATS(m_rpmask_mask, r2, m_sqr_rpmask, _CMP_GE_OQ);
if(m_mask_left == 0) {
continue;
}
/* Loop backwards through nbins. m_mask_left contains all the points that */
/* are less than rpmask at the beginning of the loop. */
for(int kbin=nbins-1;kbin>=1;kbin--) {
const AVX512_MASK m_bin_mask = AVX512_MASK_COMPARE_FLOATS(m_mask_left, r2,m_rupp_sqr[kbin-1],_CMP_GE_OS);
npairs[kbin] += bits_set_in_avx512_mask_DOUBLE[m_bin_mask];
/* ANDNOT(X, Y) -> NOT X AND Y */
m_mask_left = AVX512_MASK_BITWISE_AND_NOT(m_bin_mask, m_mask_left);
if(m_mask_left == 0) {
break;
}
}
}
}
}

```



# github.com/manodeep/Corrfunc/

This repository Search Pull requests Issues Gist

manodeep / Corrfunc Unwatch 2 Star 5 Fork 4

Code Issues 19 Pull requests 0 Wiki Pulse Graphs Settings

Blazing fast correlation functions on the CPU <https://manodeep.github.io/Corrfunc> — Edit

376 commits 5 branches 9 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

Commit	Message	Time
manodeep	Missed the correct volume normalization factor	Latest commit 541be2b 5 days ago
Corrfunc	Bumped to version 1.0 [ci skip]	a month ago
io	Trying to fix C formatting [ci skip]	3 months ago
paper	Added the updated pdfs and the bibliography file	4 months ago
utils	Fixed a bug in gridlink_index where the code would crash if the first...	a month ago
xi_mock	Protected the OpenMP constructs.	a month ago
xi_theory	Missed the correct volume normalization factor	5 days ago
.gitignore	Added the updated pdfs and the bibliography file	4 months ago
.travis.yml	First run make and then make install for TRAVIS. Updated default valu...	a month ago
FAQ	Update FAQ	4 months ago

Framework: PVD calculation reduced from 600+ hours to ~1 min

# github.com/manodeep/Corrfunc/

Blazing fast correlation functions on the CPU. <http://corrfunc.readthedocs.io/> — Edit

723 commits   5 branches   10 releases   1 contributor   MIT

Branch: master   New pull request   Create new file   Upload files   Find file   Clone or download

Commit	Description	Time
manodeep	Added the paper directory as to be ignored [ci skip]	Latest commit a9d601e 20 hours ago
Corrfunc	Forgot to add the max_cells_per_dim to the theory python wrappers	6 days ago
docs	Spruced up the docs.	21 days ago
io	Beginning the removal of all hard-coded constants to sizeof	a month ago
mocks	Finished updating tests to better compare floats. Fixes #94	a day ago
paper	Added the rmax scaling python script [ci skip]	15 days ago
theory	Added the per cell timings for wp. Needs to be added to all other rou...	20 hours ago
utils	Added the per cell timings for wp. Needs to be added to all other rou...	20 hours ago
.gitignore	Added the paper directory as to be ignored [ci skip]	20 hours ago
.travis.yml	Still trying to fix the imports. Removed OSX allowed failures from TR...	19 days ago
CHANGELOG.rst	Added a changelog with the history of releases [ci skip]	28 days ago
utils	Update version of the job file. Can read in arbitrary numbers of re...	2 years ago

Framework: PVD calculation reduced from 600+ hours to ~1 min  
Framework: PVD calculation reduced from 600+ hours to ~1 min

# github.com/manodeep/Corrfunc/

manodeep / Corrfunc

Unwatch 8 Unstar 32 Fork 13

Code Issues 15 Pull requests 0 Projects 0 Wiki Insights Settings

⚡⚡⚡ Blazing fast correlation functions on the CPU. <http://corrfunc.readthedocs.io/> Edit

astrophysics galaxies cosmology large-scale-structure pair-counting intrinsics python c openmp simd avx sse42

correlation-functions Manage topics

831 commits 3 branches 13 releases 5 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Commit Message	Time Ago
Corrfunc	Added Nick Hand as a core contributor. Fixed some docstrings renderin...	20 days ago
docs	Added Nick Hand as a core contributor. Fixed some docstrings renderin...	20 days ago
io	Added weights (#99)	a year ago
mocks	DD(s,mu) function for mocks/theory (#130) (#132)	21 days ago
paper	Code health fixes (#148)	a month ago
theory	DD(s,mu) function for mocks/theory (#130) (#132)	21 days ago
utils	DD(s,mu) function for mocks/theory (#130) (#132)	21 days ago
.gitignore	Added weights (#99)	a year ago
.landscape.yaml	Moved the strictness setting on landscape.io to the default (medium) ...	10 months ago

CODE\_OF\_CONDUCT.rst Add code of conduct [ci skip] 3 months ago

Framework: PVD calculation reduced from 600+ hours to ~1 min  
Framework: PVD calculation reduced from 600+ hours to ~1 min

# github.com/manodeep/Corrfunc/

manodeep / Corrfunc

Unwatch 8 Unstar 32 Fork 13

Code Issues 15 Pull requests 0 Projects 0 Wiki Insights Settings

⚡⚡⚡ Blazing fast correlation functions on the CPU. <http://corrfunc.readthedocs.io/> Edit

astrophysics galaxies cosmology large-scale-structure pair-counting intrinsics python c openmp simd avx sse42

correlation-functions Manage topics

831 commits 3 branches 13 releases 5 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Commit Message	Time Ago
manodeep	Added Nick Hand as a core contributor. Fixed some docstrings renderin...	Latest commit 6ea5c76 20 days ago
Corrfunc	Added Nick Hand as a core contributor. Fixed some docstrings renderin...	20 days ago
docs	Added Nick Hand as a core contributor. Fixed some docstrings renderin...	20 days ago
io	Added weights (#99)	a year ago
mocks	DD(s,mu) function for mocks/theory (#130) (#132)	21 days ago
paper	Code health fixes (#148)	a month ago
theory	DD(s,mu) function for mocks/theory (#130) (#132)	21 days ago
utils	DD(s,mu) function for mocks/theory (#130) (#132)	21 days ago
.gitignore	Added weights (#99)	a year ago
.landscape.yaml	Moved the strictness setting on landscape.io to the default (medium) ...	10 months ago
CODE_OF_CONDUCT.rst	Add code of conduct [ci skip]	3 months ago

Framework: PVD calculation reduced from 600+ hours to ~1 min  
Framework: PVD calculation reduced from 600+ hours to ~1 min

# Memory access is slow

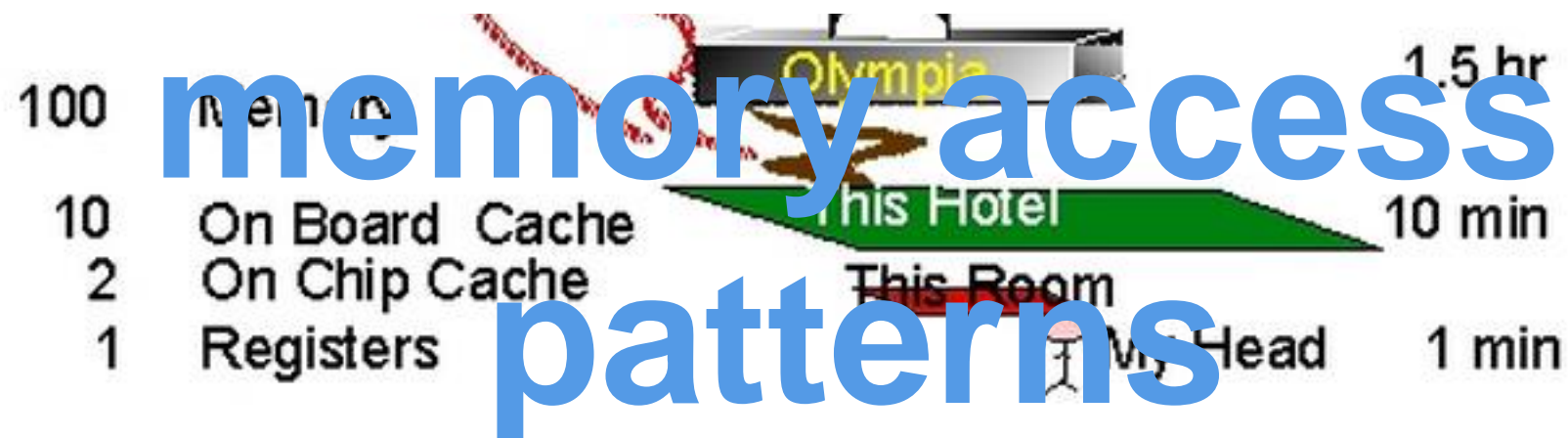
- Speed of light limitations (30 cm/ns)
  - For a 3 GHz clock, light only travels 10 cm
- Many hardware layers between requesting memory and getting data
- CPUs need to perform many calculations simultaneously

# Bottlenecks in computing

<https://blog.codinghorror.com/the-infinite-space-between-words/>

depends on  
memory access  
patterns

How fast code runs  
depends on  
memory access  
patterns





# How a CPU keeps busy (also why we got “Meltdown”)

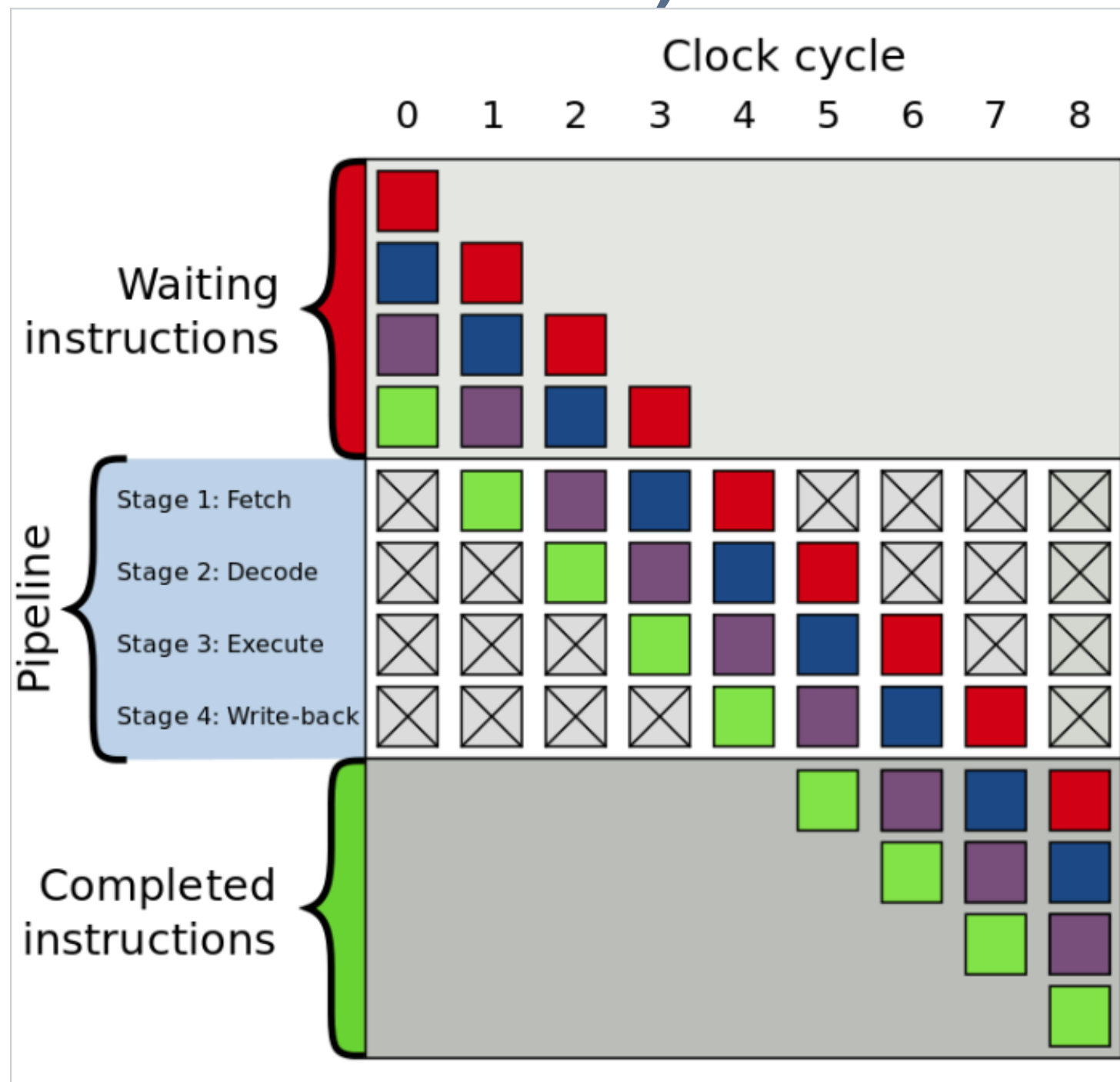
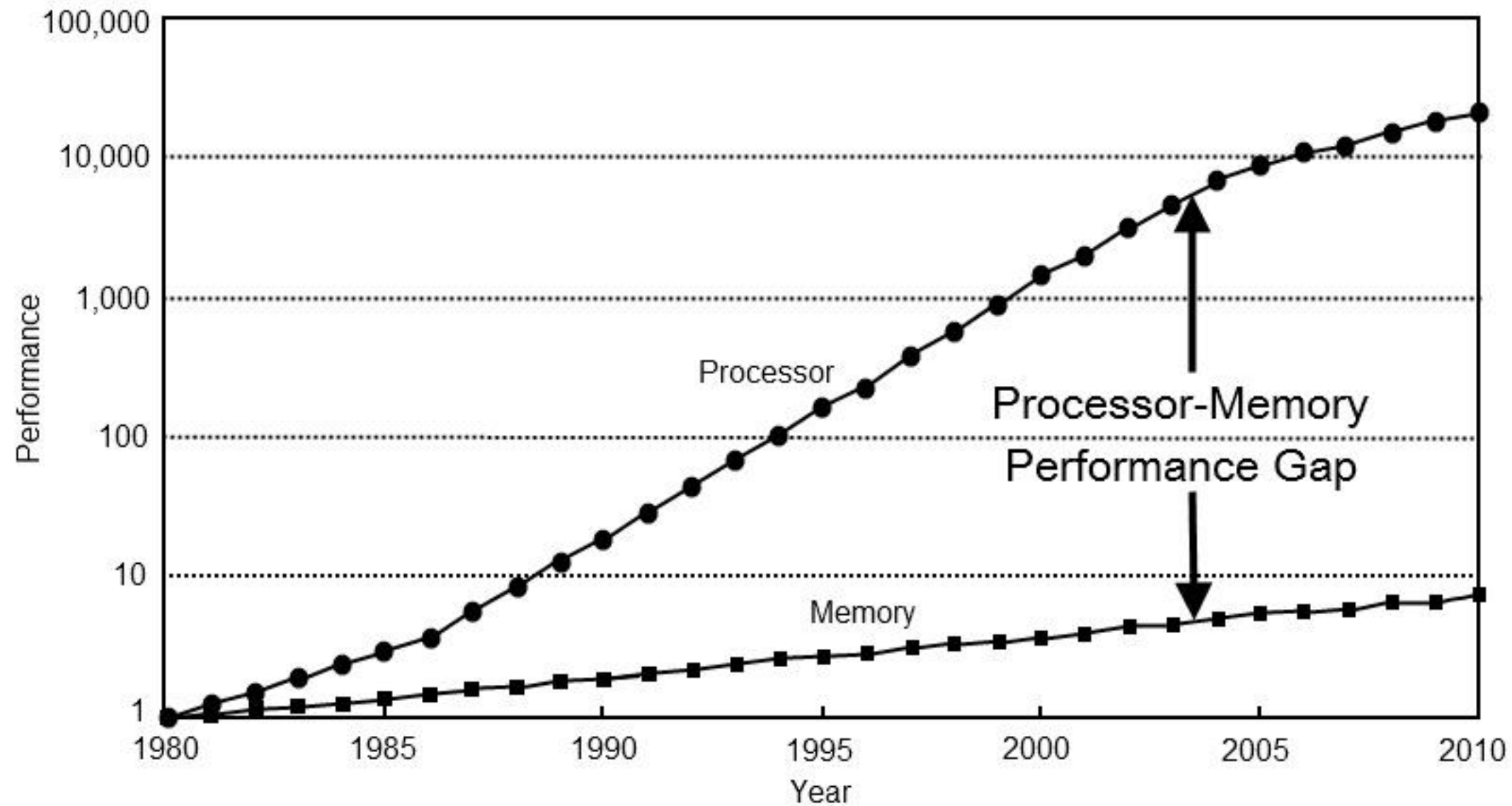


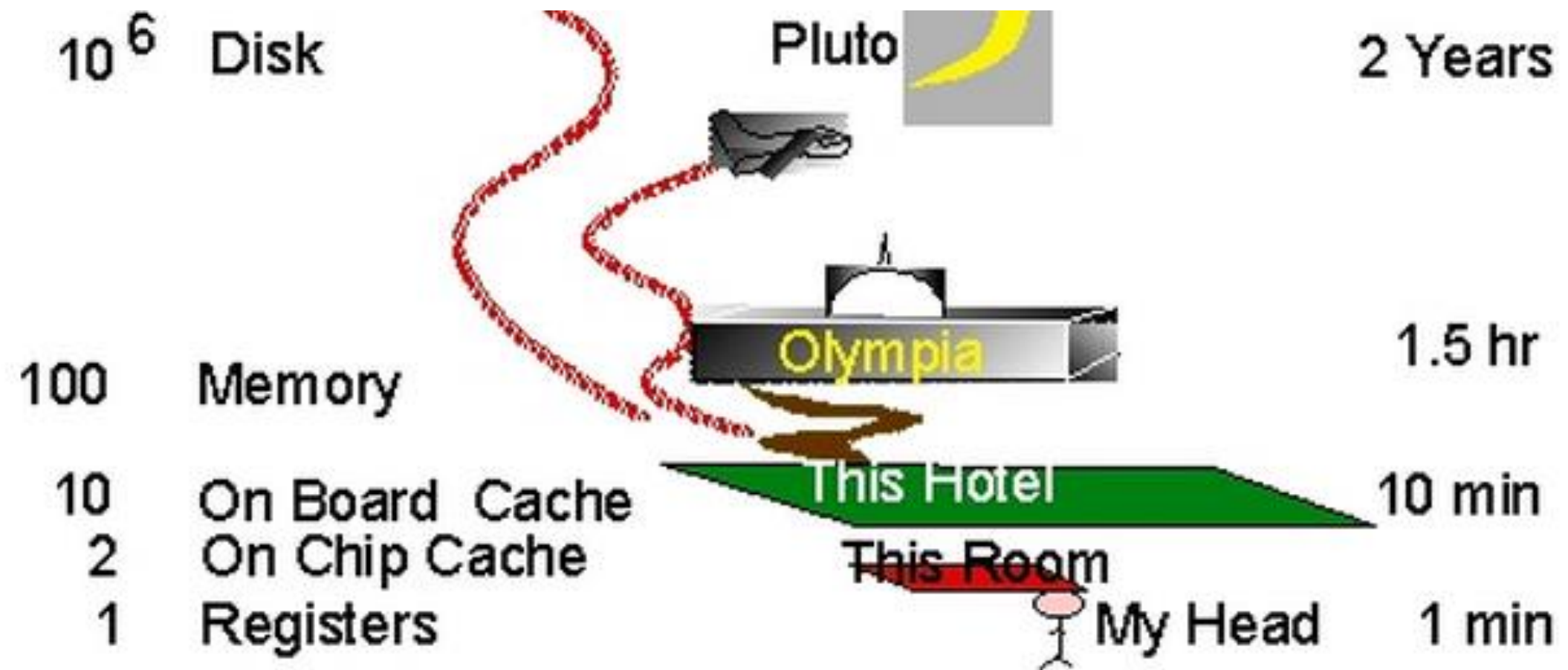
Figure 1: Example of 4-stage pipeline. The colored boxes represent instructions independent of each other

# CPU Performance Bottlenecks



# Memory access is slow

<https://blog.codinghorror.com/the-infinite-space-between-words/>



Memory access is slow

depends on

memory access

patterns

How fast code runs

depends on

memory access

patterns

<https://blog.codinghorror.com/the-infinite-space-between-words/>

10<sup>6</sup>

hrs

100

hr

10

in

2

1

Registers

My Head

1 min

23

# Not all operations are equal

- Modern cpus are extremely complex and try to predict data access patterns (hence, MELTDOWN, SPECTRE hardware bugs)
  - avoid if conditions (use `?` for the ternary operator)
- Divisions are 5 times more time-consuming than multiplication
  - Beware of trigonometric functions (use trig. identities, if possible)
- Profile your code. I am wrong > 50% of the time

---

# Users vs CPU vendor

---

- **User:** Fastest time to solution is better
- **Vendor:** Lowest power consumption for a fixed problem size while maintaining/improving time to solution
- These two metrics **are not the same**



---

# User wishlist vs CPU Features

---

- **User:** Faster CPU clocks (write old-style code)
- Physics is a buzzkill
- **Vendor:** Slower and many more individual cores per cpu (think GPUs) with wider vector widths (more calculations per clock tick)