An Automated Machine Learning Approach for Data Locality Optimizations in Chapel

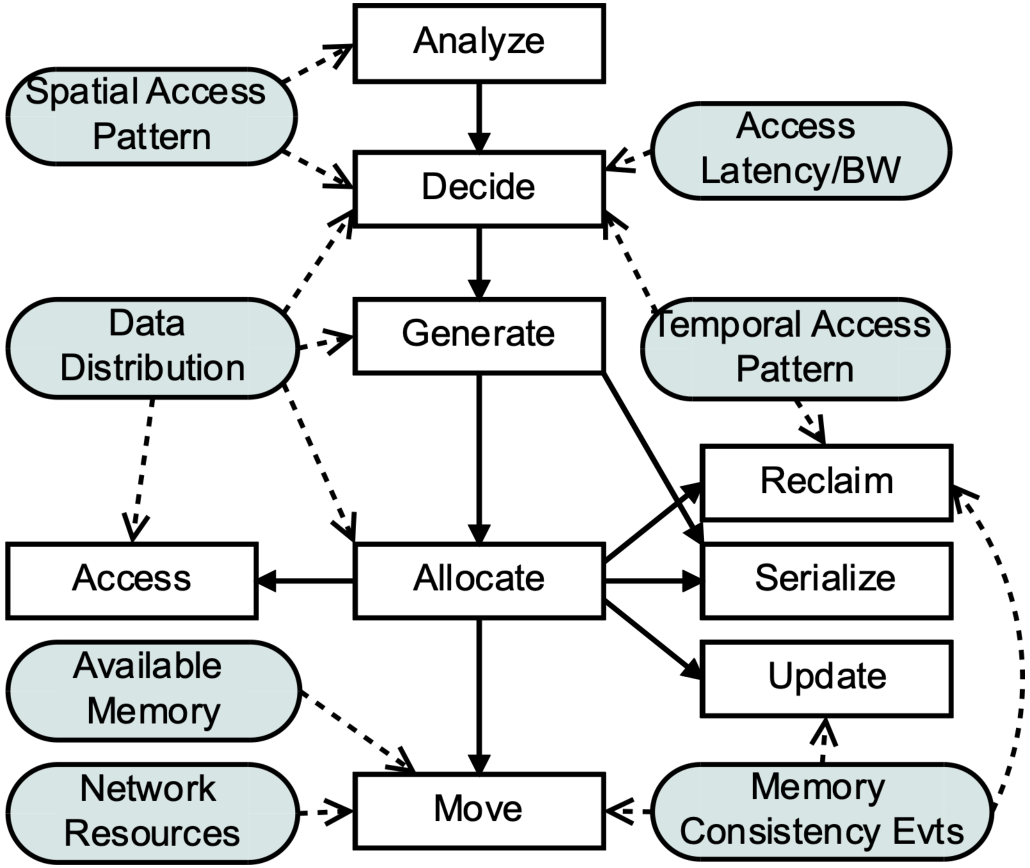
Engin Kayraklioglu
Hewlett Packard Enterprise*

Tarek El-Ghazawi
The George Washington University

This talk is a summary of author's Ph.D. dissertation he completed before joining Cray/HPE

Data Locality Optimizations in HPC

- Data locality optimization is complicated
- Many sub-tasks…
- … relying on
  - system characteristics
  - application characteristics



- Existing approaches
  - Programmer does everything
  - Language ~~does~~ *tries to do* everything

- This talk focuses on aggregated communication

Optimizing Matrix Transpose in Chapel

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

😡 More code

```
coforall l in Locales do on l {
  var locIdxs = B.localSubdomain();
  var tDom = {locIdxs.dim(2), locIdxs.dim(1)};
  var localA: [tDom] real;
  for i in 1..numIter {
    localA = A[transposeDom];
    forall (i, j) in bLocSubDom do
      B[i,j] = localA[j, i];
    forall (i, j) in A.localSubdomain() do
      A[i,j] += 1.0;
  }
}
```

Optimizing Matrix Transpose in Chapel

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

```
coforall l in Locales do on l {
  var locIdxs = B.localSubdomain();
  var tDom = {locIdxs.dim(2), locIdxs.dim(1)};
  var localA: [tDom] real;
  for i in 1..numIter {
    localA = A[transposeDom];
    forall (i, j) in bLocSubDom do
      B[i,j] = localA[j, i];
    forall (i, j) in A.localSubdomain() do
      A[i,j] += 1.0;
  }
}
```

😡 More code

😡 More complicated code

Optimizing Matrix Transpose in Chapel

```

for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

```
coforall l in Locales do on l {
  var locIdxs = B.localSubdomain();
  var tDom = {locIdxs.dim(2), locIdxs.dim(1)};
  var localA: [tDom] real;
  for i in 1..numIter {
    localA = A[transposeDom];
    forall (i, j) in bLocSubDom do
      B[i,j] = localA[j, i];
    forall (i, j) in A.localSubdomain() do
      A[i,j] += 1.0;
  }
}
```

😡 More code

😡 More complicated code

😡 Different application logic

This talk

- … will discuss three ideas
  - a collaborative language feature for data locality optimization
  - a high-level profiler to analyze accesses to distributed arrays
  - a machine-learning based framework for complete automation

- … with specific focus on …
  - what they mean for a Chapel user

- … while mostly handwaving about …
  - implementation details
  - experimental results

Locality Aware Productive Prefetching Support (LAPPS)

- It is hard for the compiler to do locality optimization
  - Static analysis is difficult
  - Code modification is "scary"
- It is hard for the runtime to do locality optimization
  - Dynamic analysis has costs
  - They have limited view of the application
- What if the user tells them exactly how the data is accessed?

E. Kayraklioglu et al. "Locality-Aware Productive Prefetching Support for PGAS", ACM Transactions on Architecture and Code Optimizations, Vol 15, Issue 3. 2018

Matrix Transpose with LAPPS

```

for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

😡 More code

😡 More complicated code

😡 Different application logic

```
coforall l in Locales do on l {
  var locIdxs = B.localSubdomain();
  var tDom = {locIdxs.dim(2), locIdxs.dim(1)};
  var localA: [tDom] real;
  for i in 1..numIter {
    localA = A[transposeDom];
    forall (i, j) in bLocSubDom do
      B[i,j] = localA[j, i];
    forall (i, j) in A.localSubdomain() do
      A[i,j] += 1.0;
  }
}
```

Matrix Transpose with LAPPS

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

```
coforall l in Locales do on l {
  var locIdxs = B.localSubdomain();
  var tDom = {locIdxs.dim(2), locIdxs.dim(1)};
  var localA: [tDom] real;
  for i in 1..numIter {
    localA = A[transposeDom];
    forall (i, j) in bLocSubDom do
      B[i,j] = localA[j, i];
    forall (i, j) in A.localSubdomain() do
      A[i,j] += 1.0;
  }
}
```

😡 More code

😡 More complicated code

😡 Different application logic

Matrix Transpose with LAPPS

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

```
coforal
  var
  var
  var
  for
    lo
    fo

    forall (i, j) in A.localSubdomain() do
      A[i,j] += 1.0;
  }
}
```

```
A.transposePrefetch();
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

😡 More code

😡 More complicated code

😡 Different application logic

Matrix Transpose with LAPPS

🙂 Trivial code modifications

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

😡 More code

😡 More complicated code

😡 Different application logic

```
A.transposePrefetch();
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

Matrix Transpose with LAPPS

🙂 Trivial code modifications

🙂 Application logic remains identical

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

😡 More code

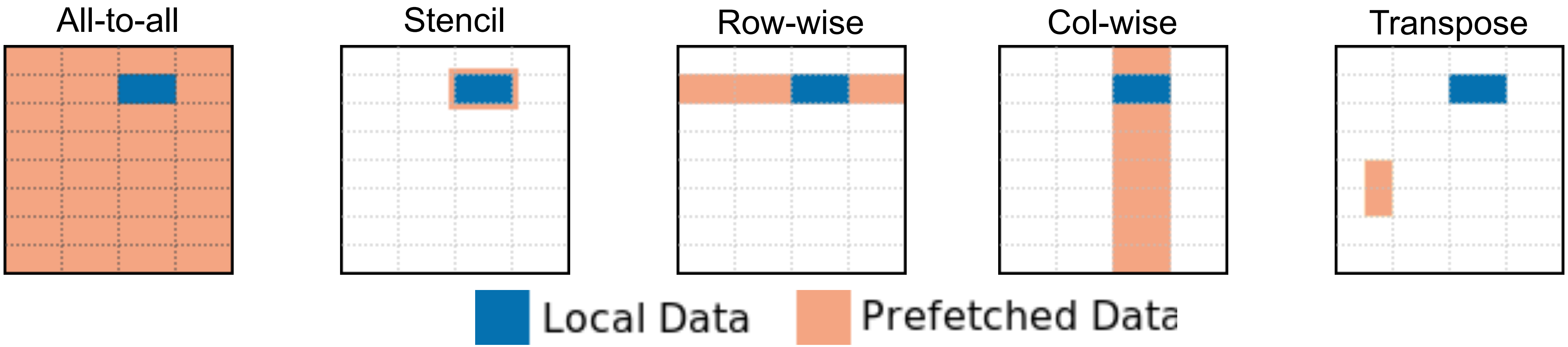
😡 More complicated code

😡 Different application logic

```
A.transposePrefetch();
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

```
cofora
  var
  var
  var
  for
      A[i,j] += 1.0;
  }
}
```

Prefetch Patterns



- Also a "Custom" pattern
  - User gives an array to describe what data is needed by each locale
  - Figuring out the communication is still handled by the library/runtime
  - **The custom pattern can also be used by automatic code generators**

LAPPS Experiments Summary

- Performance tested with synthetic and application benchmarks
- **Good strong and weak scaling** performance
- **Up to two orders of magnitude faster** than un-optimized
- **On-par** with manually optimized application
- **Negligible memory footprint** increase over manually-optimized

LAPPS is good, but…

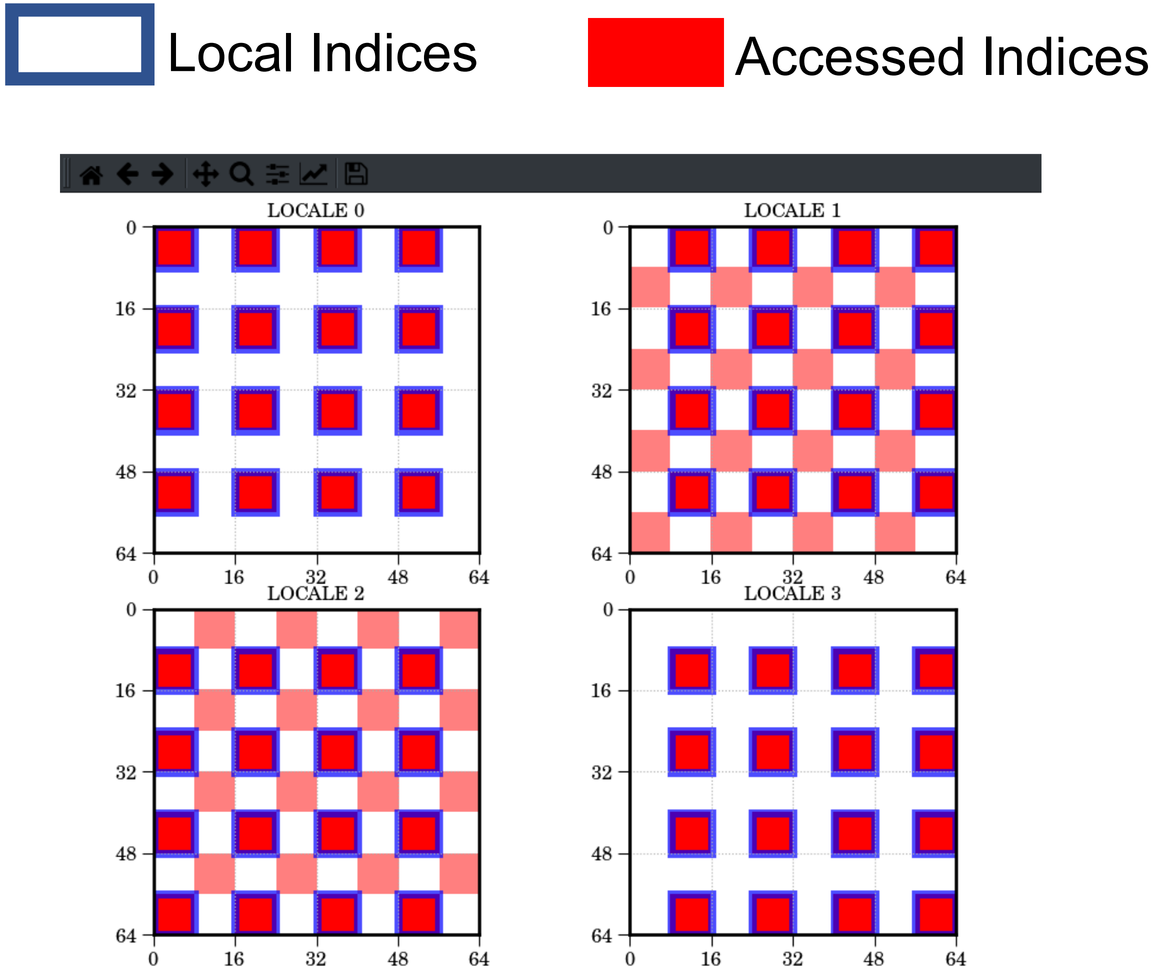
…still relies on the programmer **understanding** the data access patterns and **making** the correct prefetch call

Can a high-level, data-centric application profiler help the programmer use LAPPS?

E.Kayraklioglu et al. "An Access Pattern Analysis Tool for Distributed Arrays", ACM Computing Frontiers 2018

Access Pattern Analysis Tool (APAT)

- A profiler that is
  - high-level
  - data-centric
- And that does
  - Collect accessed indices
  - Help identify spatial patterns
- And that can be used
  - standalone,
  - or with LAPPS



A screenshot from the GUI
HPCC-PTRANS with 4 locales

APAT helps programmers use LAPPS, but…

…interpreting the APAT output and appropriately using LAPPS is still the programmer's duty.

Can we automate the process by making AI learn the access patterns and optimize the application and using LAPPS?

E.Kayraklioglu et al. "A Machine Learning Approach for Productive Data Locality Exploitation in Parallel Computing Systems", IEEE/ACM CCGrid 2018

E.Kayraklioglu et al. "A Machine-Learning-Based Framework for Productive Locality Exploitation", IEEE TPDS, under review

Matrix Transpose with LAPPS

🙂 **Trivial code modifications**

🙂 **Application logic remains identical**

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

😡 **More code**

😡 **More complicated code**

😡 **Different application logic**

```
cofora
  var                                              1)};
  var
  var
  for
    lo
    fo

    forall (i, j) in A.localSubdomain() do
      A[i,j] += 1.0;
  }
}
```

```
A.transposePrefetch();
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

Matrix Transpose with LAPPS

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

🙂 Trivial code modifications

🙂 Application logic remains identical

😡 More code

😡 More complicated code

😡 Different application logic

```
coforall l in Locales do on l {
  var lo
  var tD
  var lo
  for i
    loca
    fora
      B[
    fora
      A[i,j] += 1.0;
  }
}
```

```
pragma optimize arrays(A)
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

Matrix Transpose with LAPPS

```
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

😡 More code

😡 More complicated code

😡 Different application logic

🙂 Trivial code modifications

🙂 Application logic remains identical

🙂 No need to worry about pattern

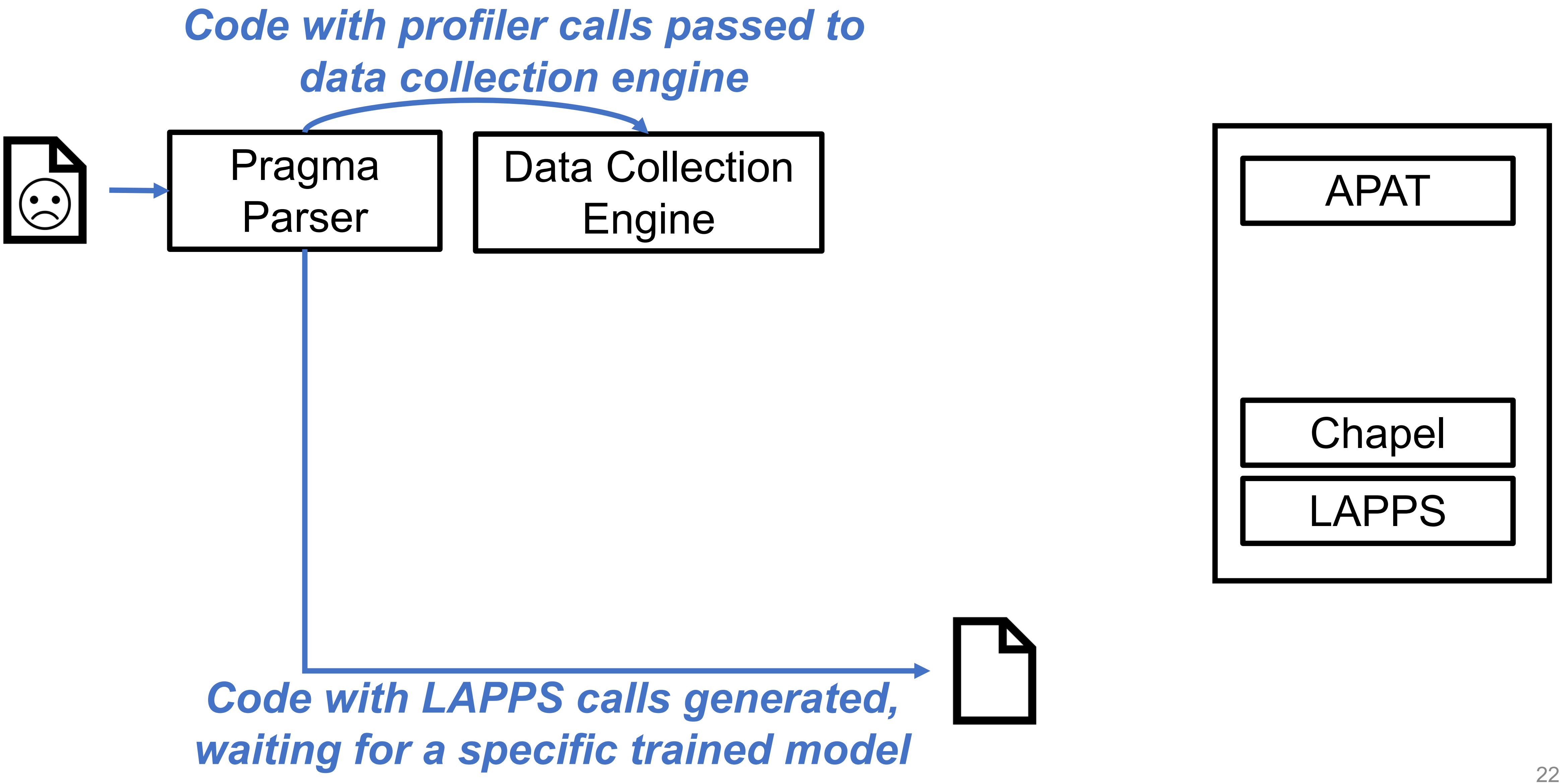
```
coforall l in Locales do on l {
  var lo
  var t[
  var lo
  for i
    loca
    fora
      B[
    fora
      A[i,j] += 1.0;
  }
}
```

```
pragma optimize arrays(A)
for i in 1..numIter {
  forall (i, j) in B.domain do
    B[i, j] = A[j, i];
  forall a in A do
    a += 1.0;
}
```

How it works…

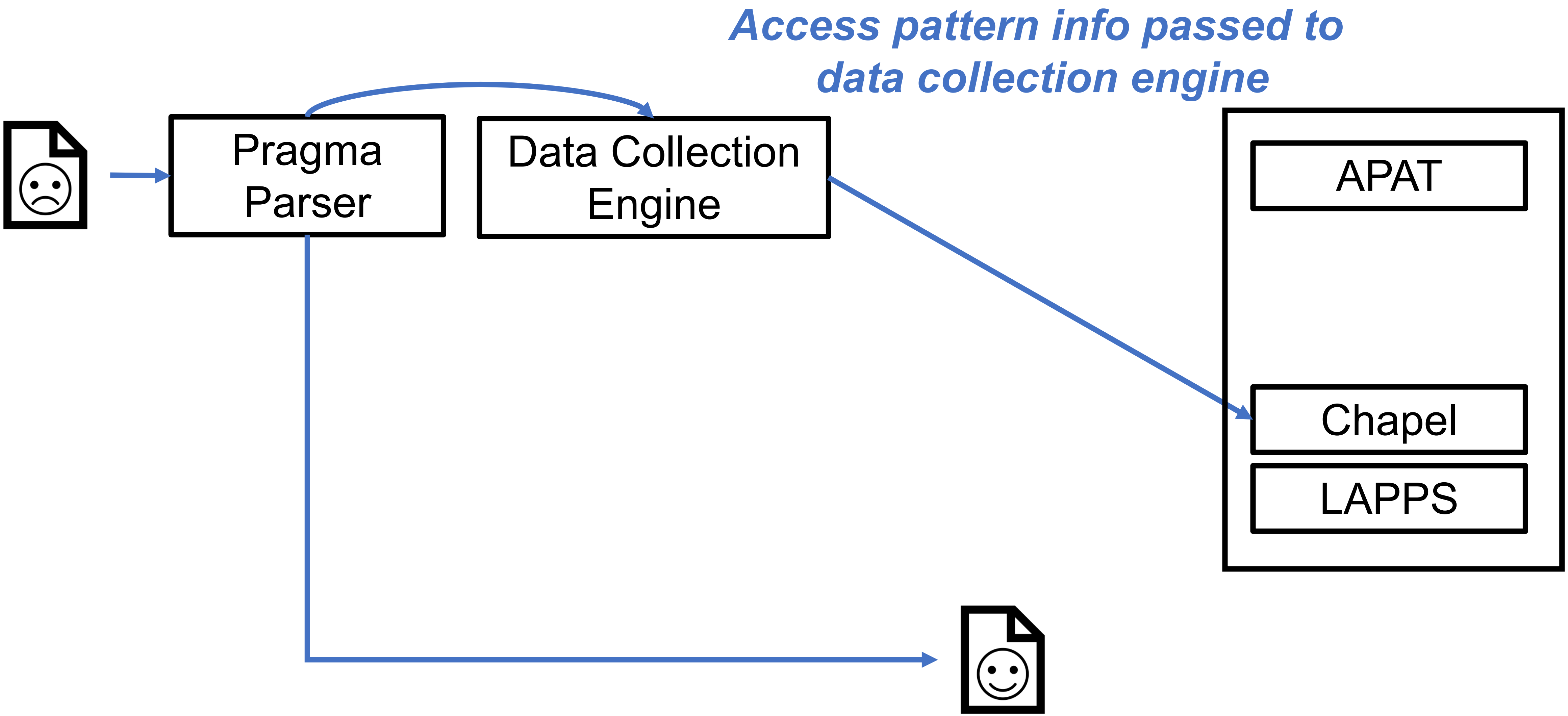
User gives code with pragmas

How it works…



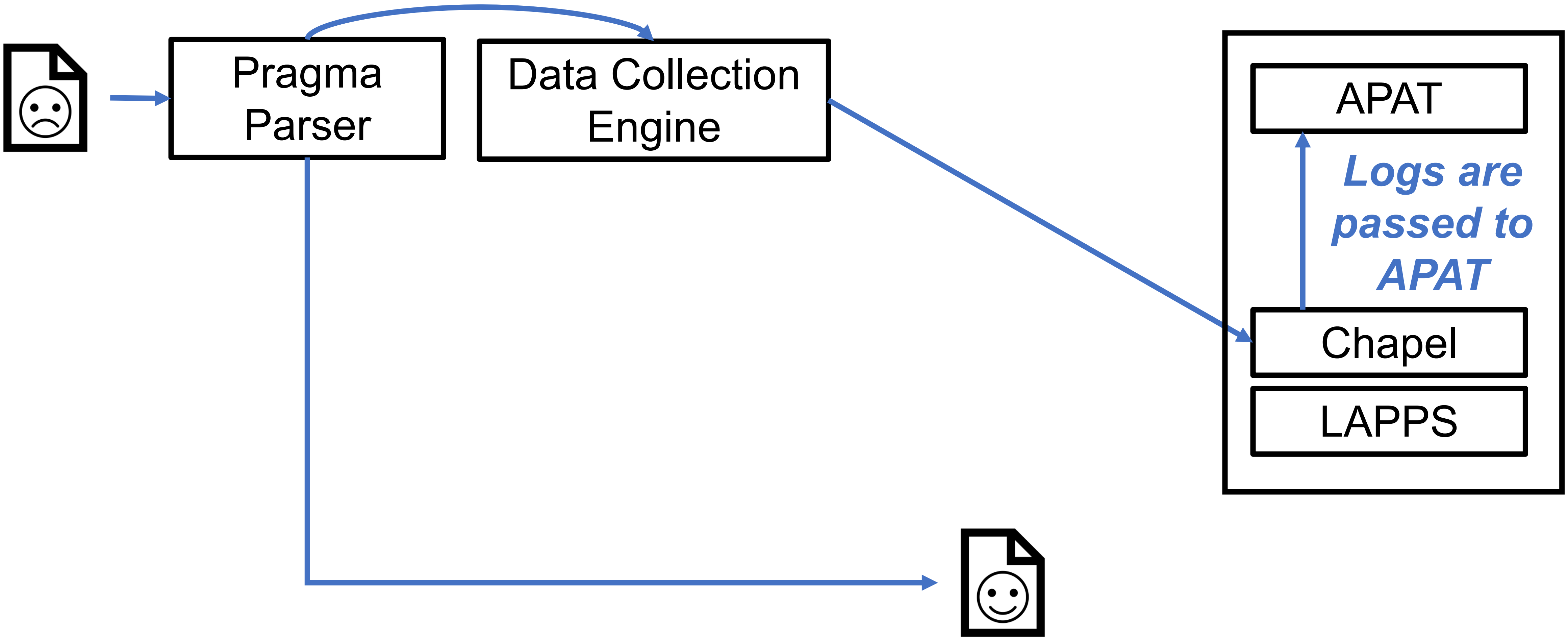
Code with profiler calls passed to data collection engine
Pragma Parser
Data Collection Engine
APAT
Chapel
LAPPS
Code with LAPPS calls generated, waiting for a specific trained model

How it works…



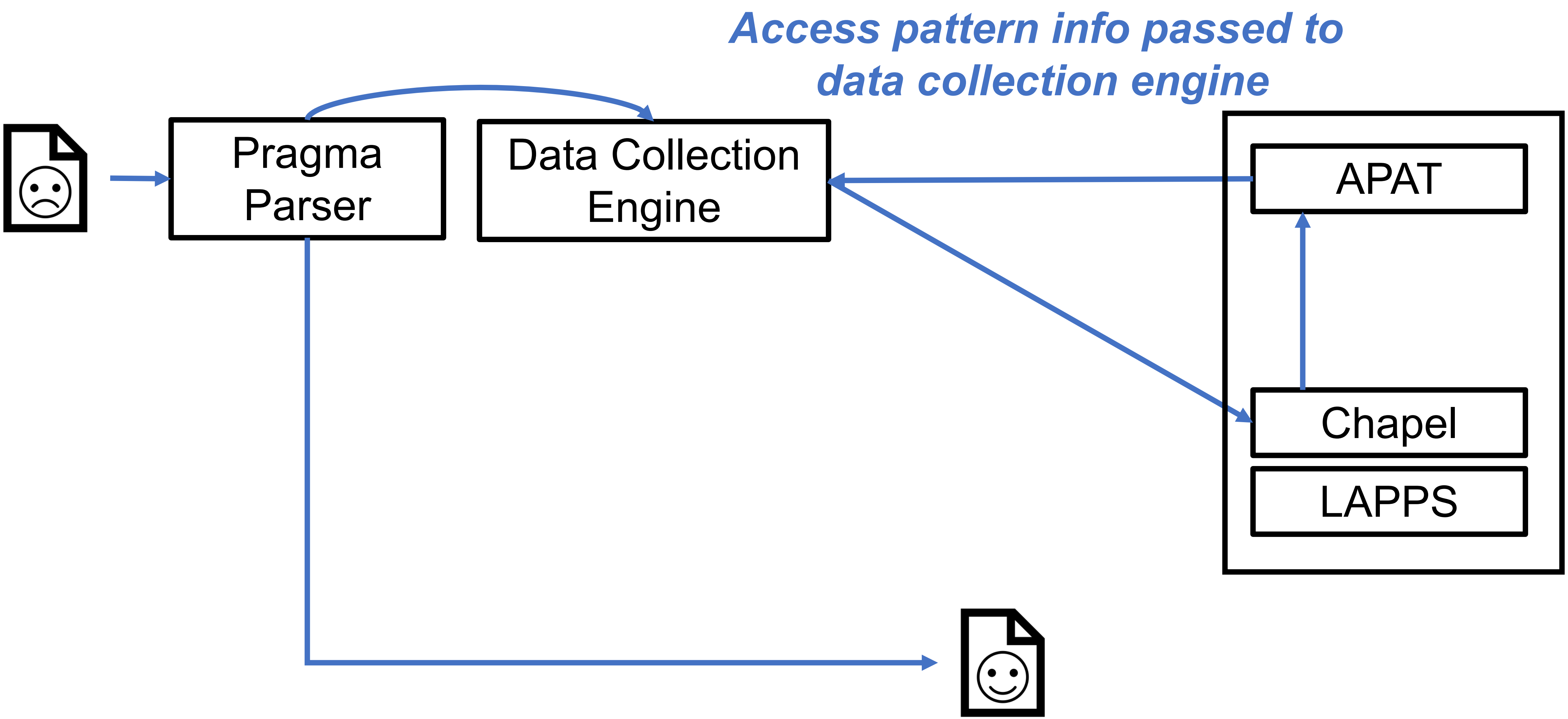
Access pattern info passed to data collection engine
Pragma Parser
Data Collection Engine
APAT
Chapel
LAPPS

How it works…



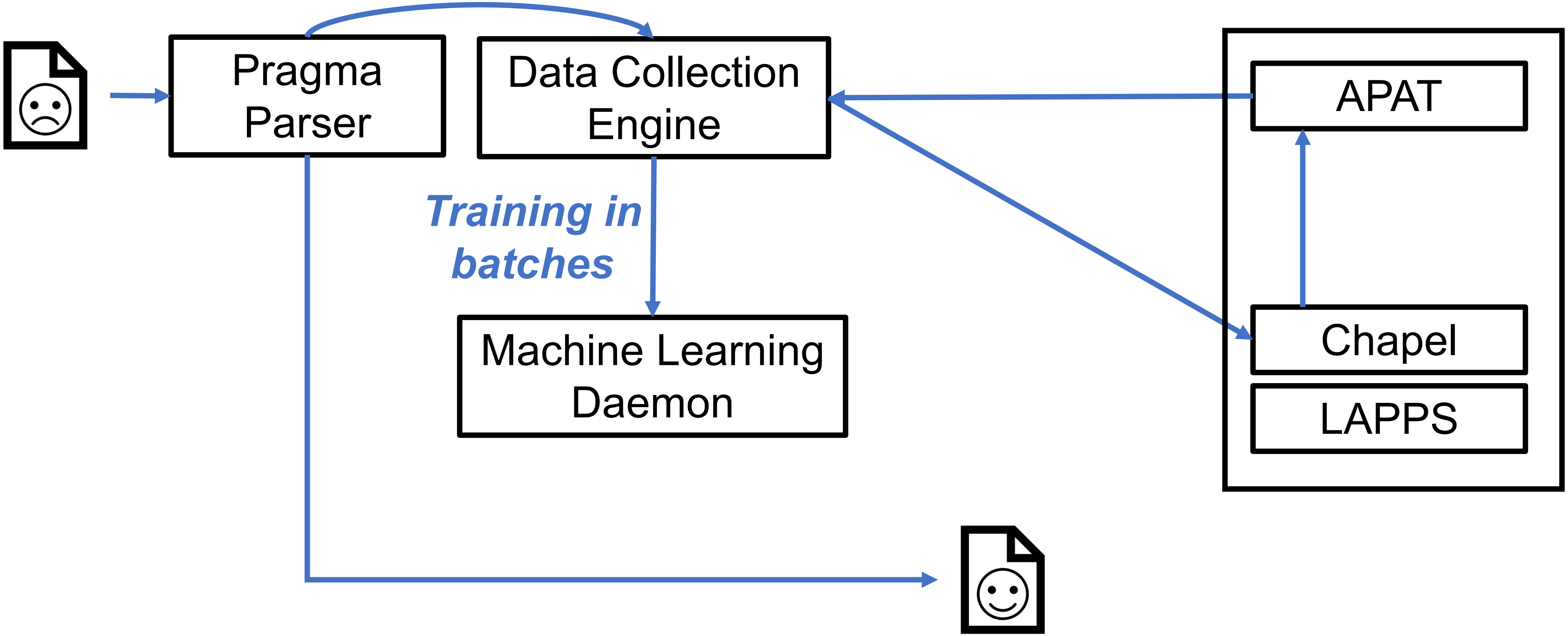
Pragma Parser
Data Collection Engine
APAT
Logs are passed to APAT
Chapel
LAPPS

How it works…



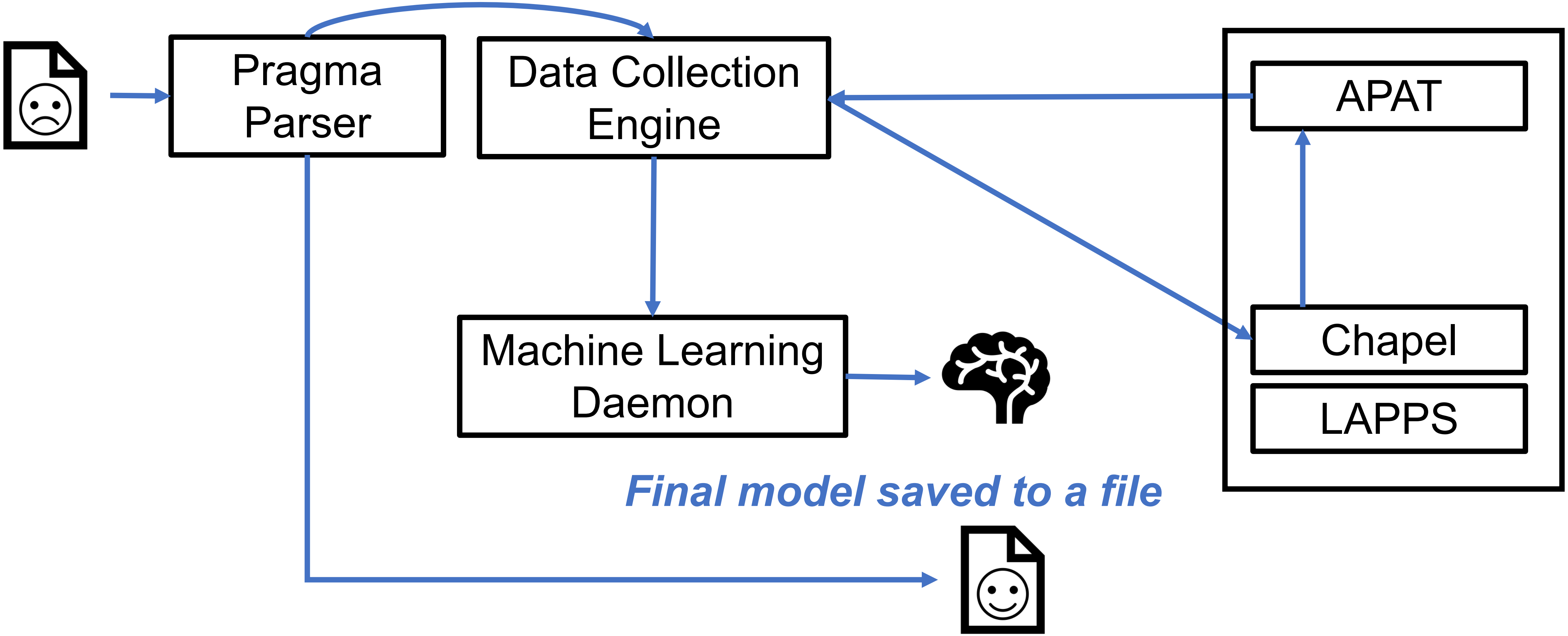
Access pattern info passed to data collection engine
Pragma Parser
Data Collection Engine
APAT
Chapel
LAPPS

How it works…

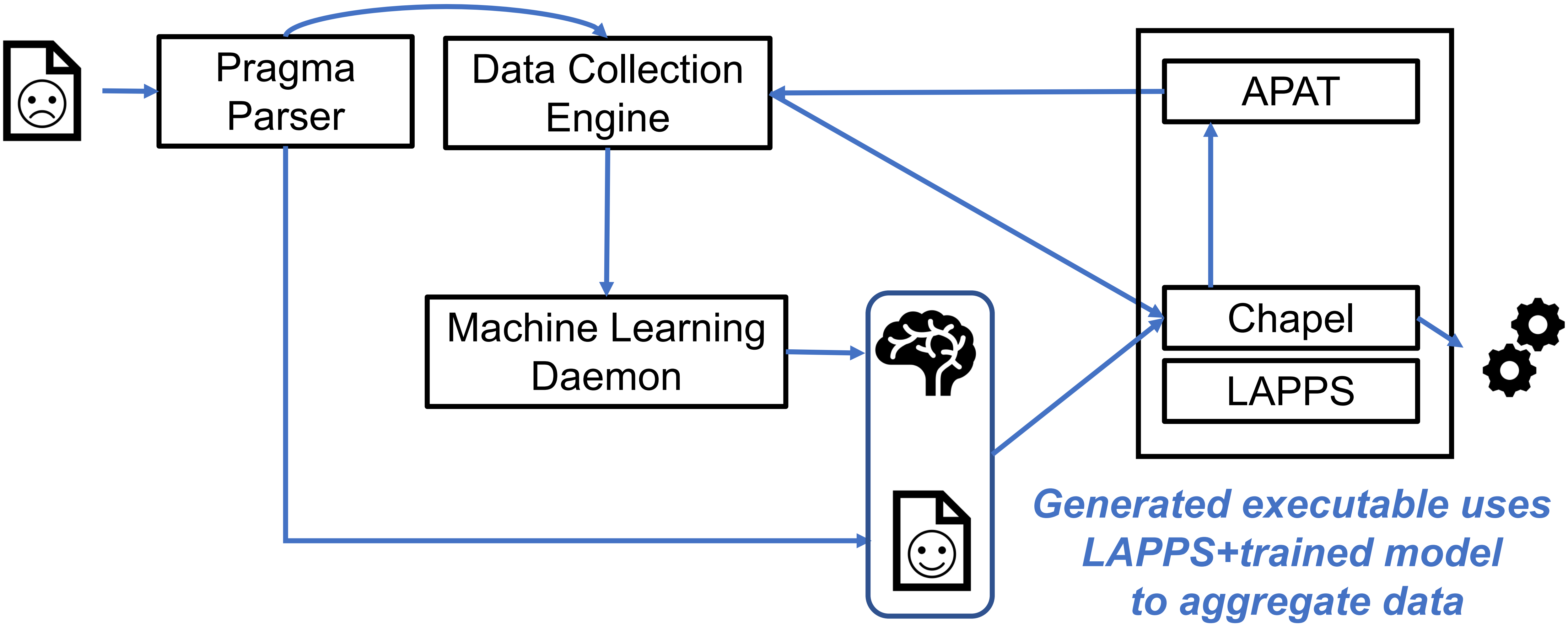


Pragma Parser
Data Collection Engine
Training in batches
Machine Learning Daemon
APAT
Chapel
LAPPS

How it works…


Pragma Parser
Data Collection Engine
Machine Learning Daemon
APAT
Chapel
LAPPS
Final model saved to a file

How it works…

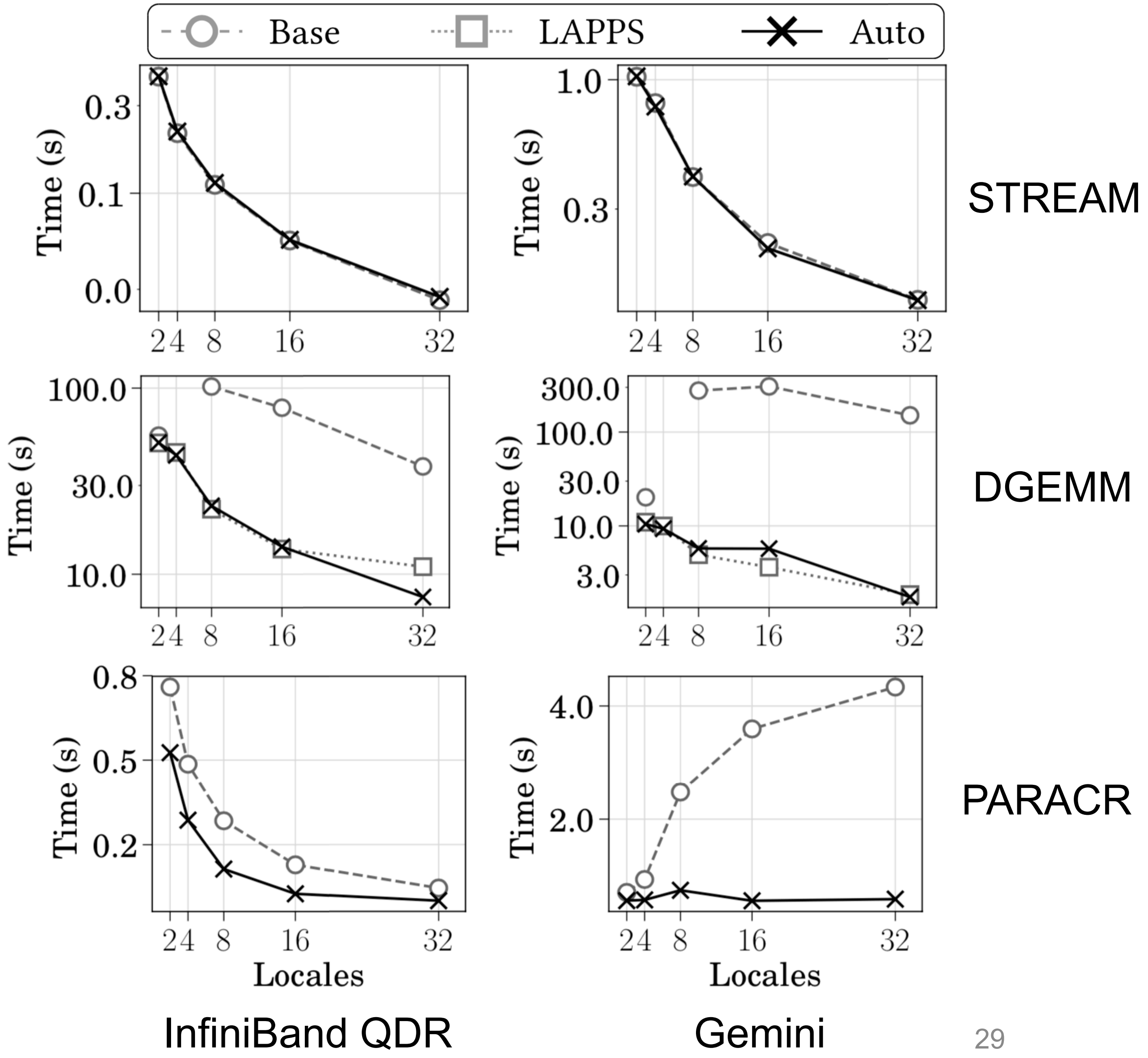


Generated executable uses LAPPS+trained model to aggregate data

Performance Results Summary

- On-par performance
- Good scalability
- Portable optimization
- Low memory footprint

- Very little programmer effort
- Short aggregate training time
  - 3 min in a 50 node cluster
  - 30 min in a personal workstation



Summary

- 3 related approaches for more productive optimizations
  - A language feature that makes coding easier
  - A profiler to help understand access patterns
  - A framework that uses machine learning to automate process

- Programmer still need to be involved
  - But in a much less disruptive fashion
  - Application correctness and performance concerns are separated
    - One person can write the application
      - without knowing/caring too much about distributed memory
    - Another can optimize it
      - without knowing/caring too much about the application

Thanks!

engin@hpe.com