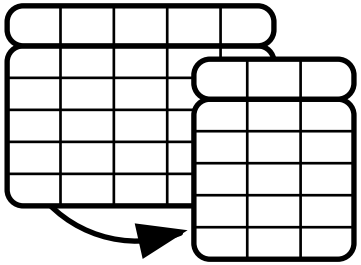
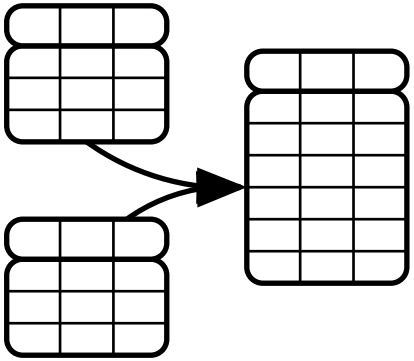
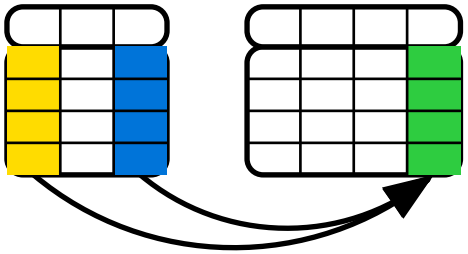
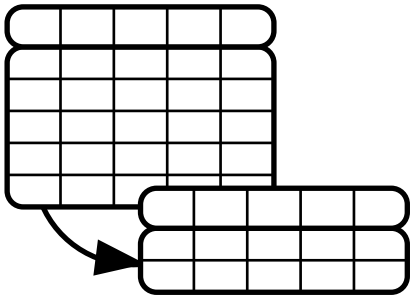
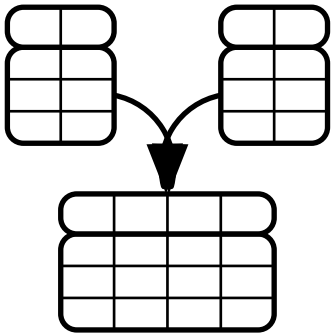
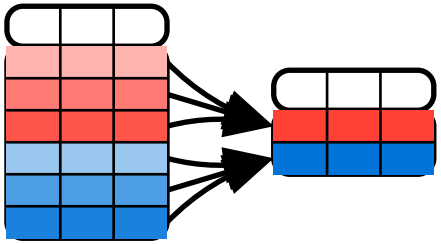
Query Evaluation Algorithms

CSE 4/562: Database Systems | Lecture 3

DB. Sys.: T.C.B.: 15.1-15.3

Recap

Desiderata

	Filter	Merge	Derive
Column			
Row			

	Filter	Merge	Derive
Column	$\pi_{A,B,\ldots}(R)$ “Project”	$R \times S$ “Cartesian Product”	$\pi_{C=A+B}(R)$ “Project”
Row	$\sigma_{A<3}(R)$ “Select” or “Filter”	$R \cup S$ “Union”	$\Sigma_{A,B=\ \mathrm{COUNT}()}(R)$ “Aggregate”

A note on join terminology...

Cartesian Product

- $R \times S$: Every pair of one tuple from R and S

Join

- $R \underset{a>b}{\bowtie} S$: Only include pairs where the predicate $a > b$ is true

Equi-Join

- $R \underset{a=b}{\bowtie} S$: A join that only uses equality predicates (e.g., $a = b \wedge c = d$)
- $R \underset{a}{\bowtie} S$: If the equi-join columns have the same name, we just write the name(s)

Natural Join

- $R \bowtie S$: If the join predicate is omitted, assume an equi-join between all columns with the same name.

Existing Systems: Summary

1. Parse the query
2. Make a Dumb™ plan
3. Come up with better plan ideas and pick one
4. Fill in the specific algorithms and data structures

Evaluating Queries

Query Evaluation: Example

Passenger	**Name**	**Train_no**	**Dest**
	Athena	48	Buffalo
	Bragi	239	Albany
	Cerberus	241	Hudson

Train	**Train_no**	**Route**
	48	Lake Shore Ltd.
	239	Emp. Service
	241	Emp. Service

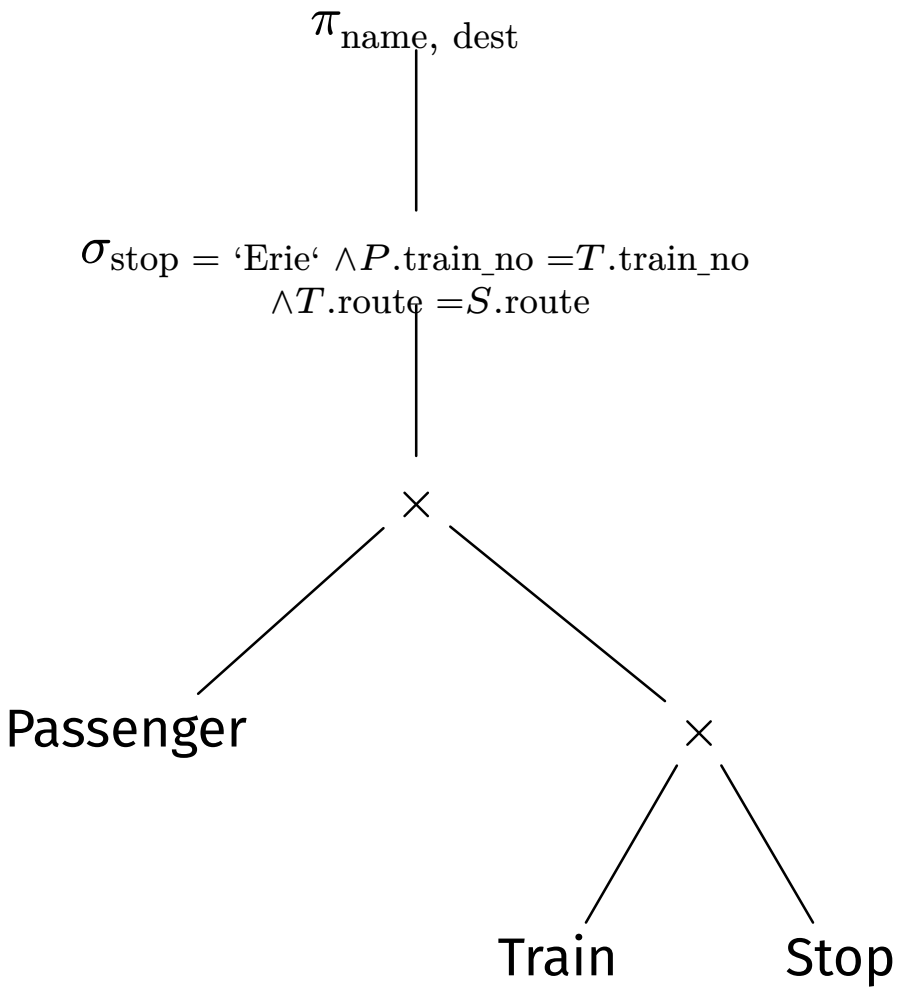
Stop	**Route**	**Stop**
	Emp. Service	NYC
	Emp. Service	Yonkers
	Emp. Service	Hudson
	Emp. Service	Albany
	Lake Shore Ltd.	Chicago
	Lake Shore Ltd.	Erie
	Lake Shore Ltd.	Buffalo

“Find me every passenger on a train that stops in Erie, and their destination”

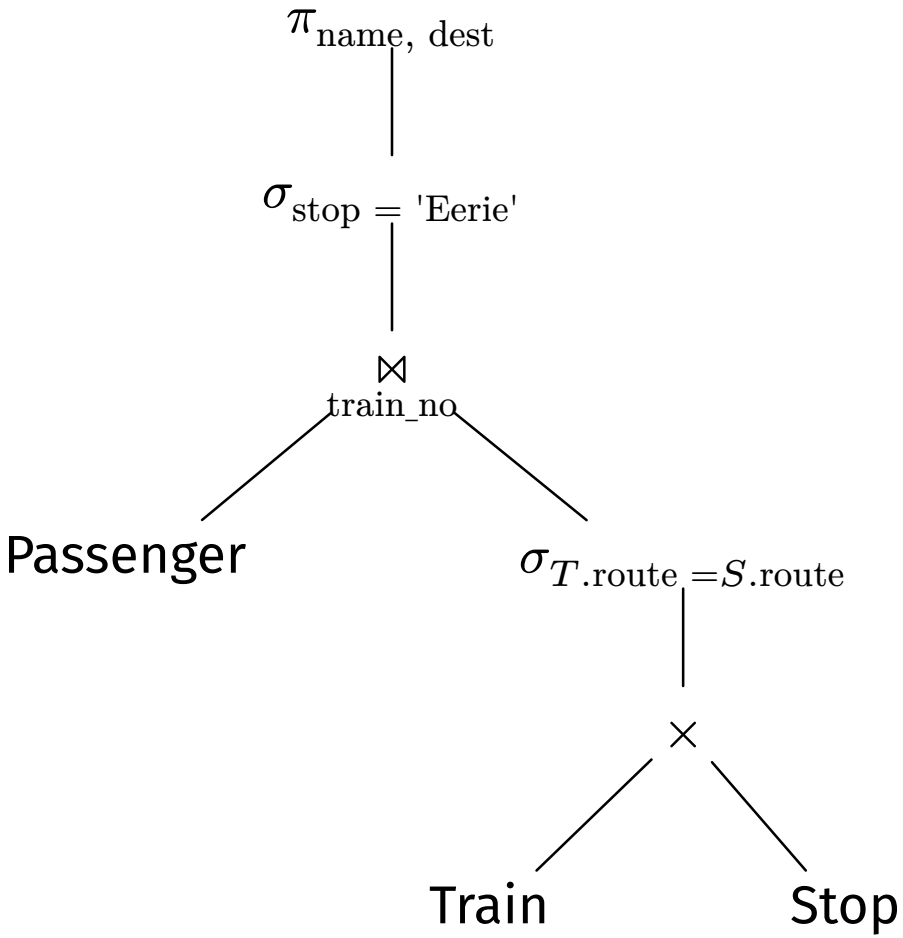
Query Evaluation: Example


```
SELECT name, dest
FROM Passenger P, Train T, Stop S
WHERE stop = 'Erie'
  AND P.train_no = T.train_no
  AND T.route = S.route
```

Query Evaluation: Example



Query Evaluation: Example



How do we evaluate this query?

Operator-at-a-Time


```
r1 = product(Train, Stop)
```

Operator-at-a-Time

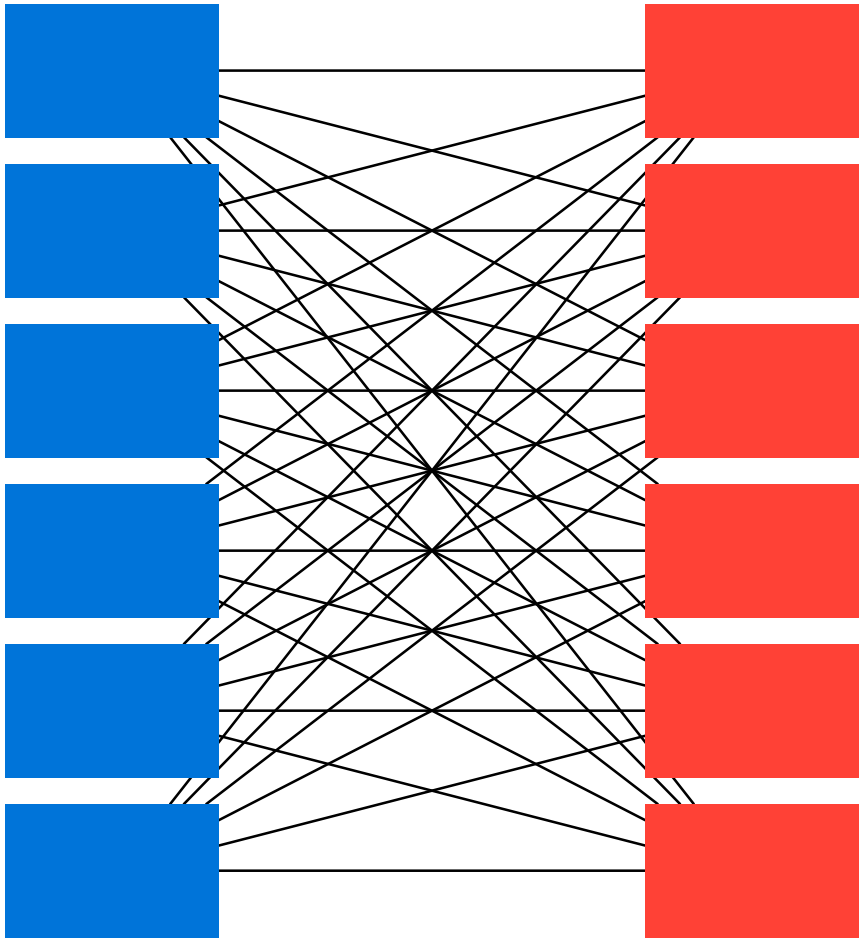
```
r1 = product(Train, Stop)
```

R_1

t_no	T.route	S.route	stop
239	Emp. Service	Emp. Service	Hudson
239	Emp. Service	Emp. Service	Albany
239	Emp. Service	Lake Shore Ltd.	Chicago
239	Emp. Service	Lake Shore Ltd.	Erie

(21 rows total)

Cartesian Product



Operator-at-a-Time


```
def product(In1, In2):
    output = []

    for r in In1:
        for s in In2:
            output += [ r + s ]

    return output
```

Operator-at-a-Time

```
r1 = product(Train, Stop)
r2 = filter(lambda row: ..., r1)
```

R_1

t_no	T.route	S.route	stop
239	Emp. Service	Emp. Service	Hudson
239	Emp. Service	Emp. Service	Albany
239	Emp. Service	Lake Shore Ltd.	Chicago
239	Emp. Service	Lake Shore Ltd.	Erie

(21 rows total)

Operator-at-a-Time

```
r1 = product(Train, Stop)
r2 = filter(lambda row: ..., r1)
```

R_2

t_no	T.route	S.route	stop
239	Emp. Service	Emp. Service	NYC
239	Emp. Service	Emp. Service	Yonkers
239	Emp. Service	Emp. Service	Hudson
239	Emp. Service	Emp. Service	Albany
241	Emp. Service	Emp. Service	NYC

(11 rows total)

Operator-at-a-Time


```
def filter(condition, In):
    output = []

    for r in In:
        if condition(r):
            output += [ r ]

    return output
```

Operator-at-a-Time

```
r1 = product(Train, Stop)
r2 = filter(lambda row: ..., r1)
r3 = join('train_no', Passenger, r2)
```

R_2

t_no	T.route	S.route	stop
239	Emp. Service	Emp. Service	NYC
239	Emp. Service	Emp. Service	Yonkers
239	Emp. Service	Emp. Service	Hudson
239	Emp. Service	Emp. Service	Albany
241	Emp. Service	Emp. Service	NYC

(11 rows total)

Operator-at-a-Time

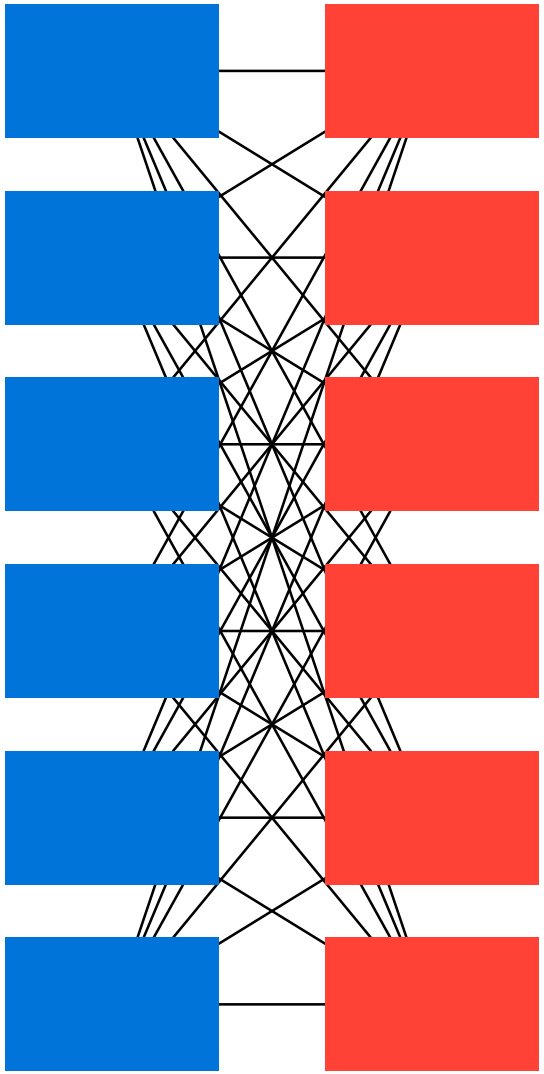
```
r1 = product(Train, Stop)
r2 = filter(lambda row: ..., r1)
r3 = join('train_no', Passenger, r2)
```

R_3

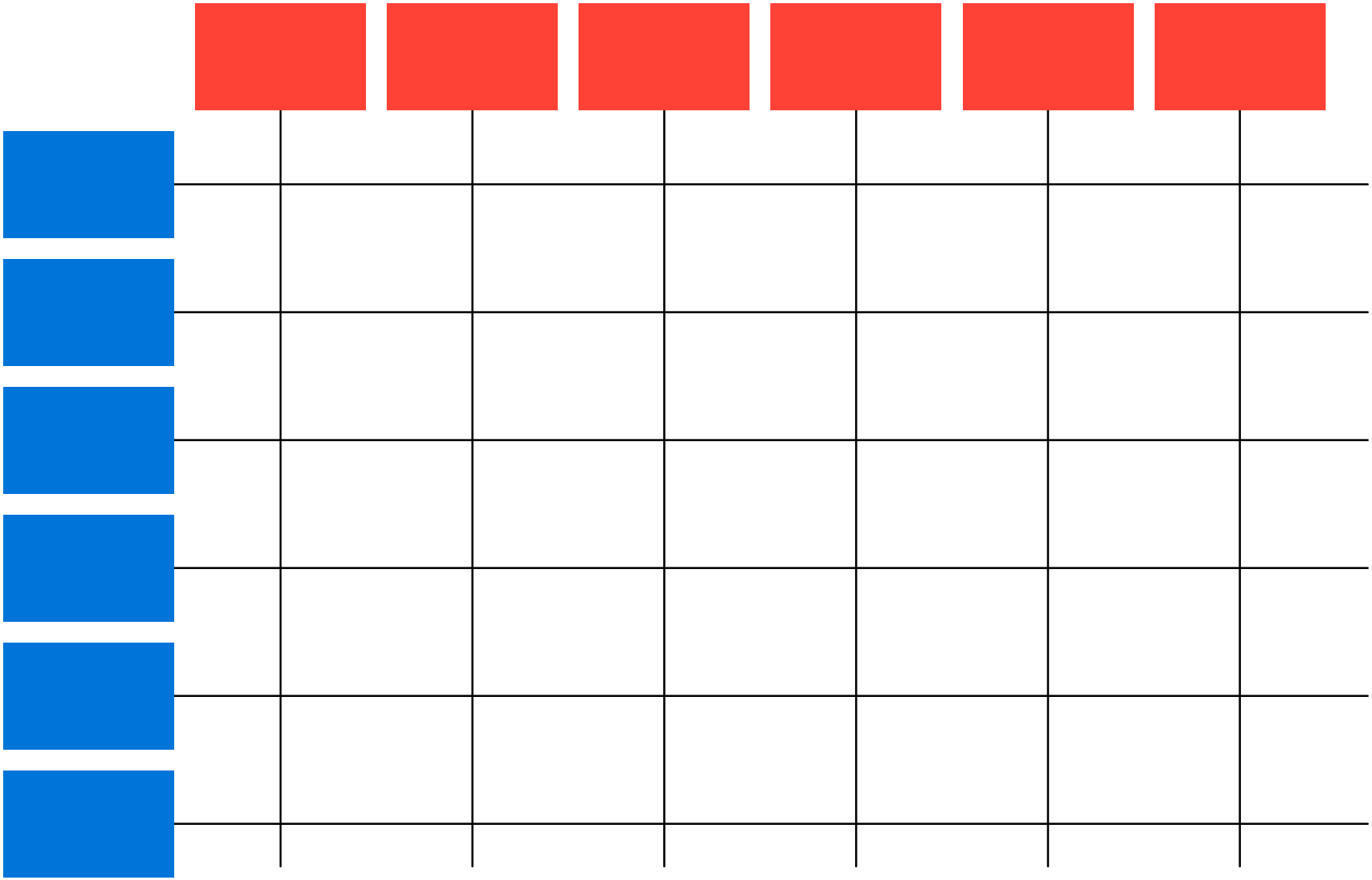
name	dest	P.t_no	T.route	stop
Athena	48	Buffalo	Lake Shore Ltd.	Erie
Athena	48	Buffalo	Lake Shore Ltd.	Buffalo
Bragi	239	Albany	Emp. Service	NYC
Bragi	239	Albany	Emp. Service	Yonkers

(11 rows total)

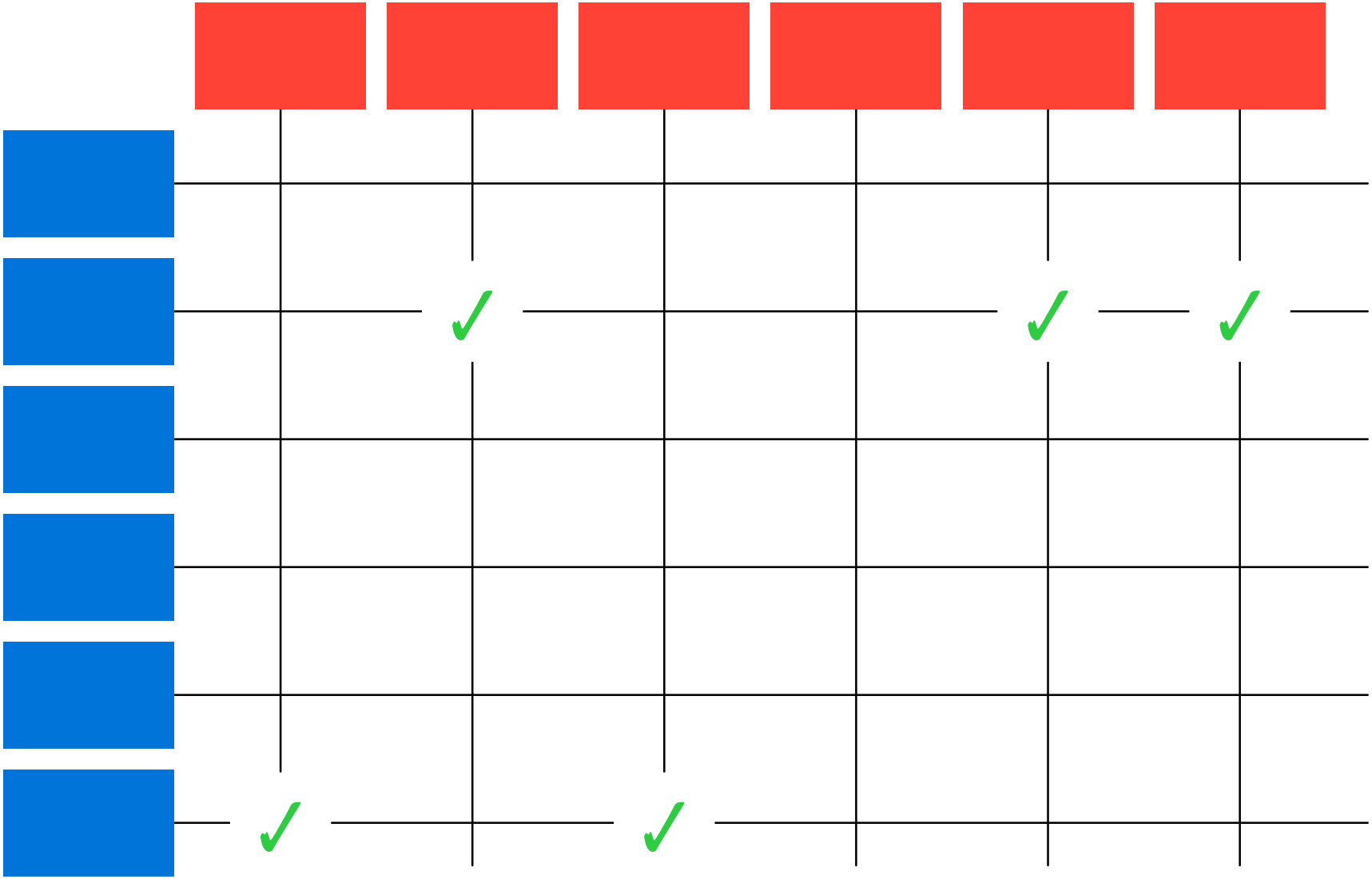
Join



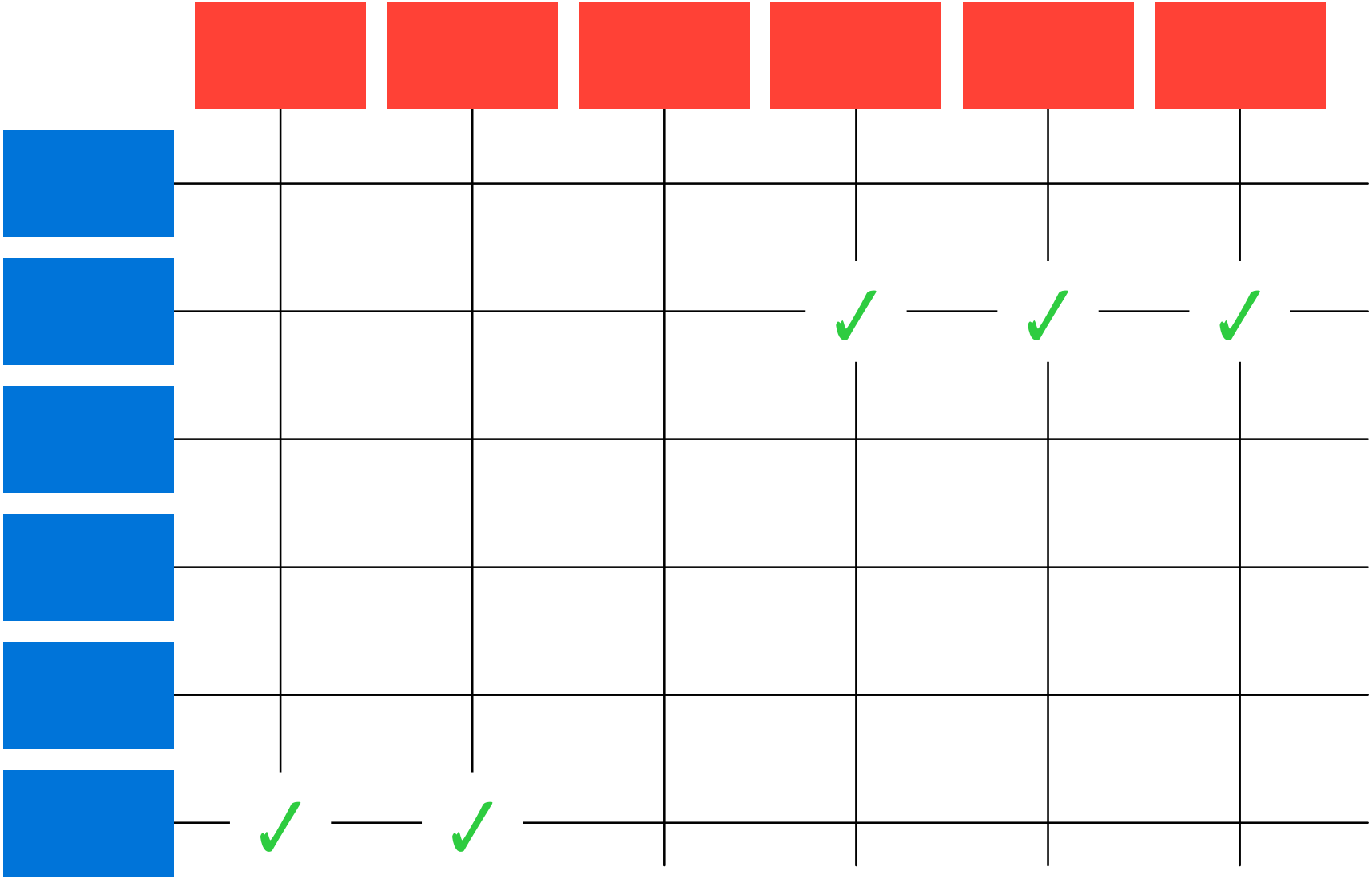
Join



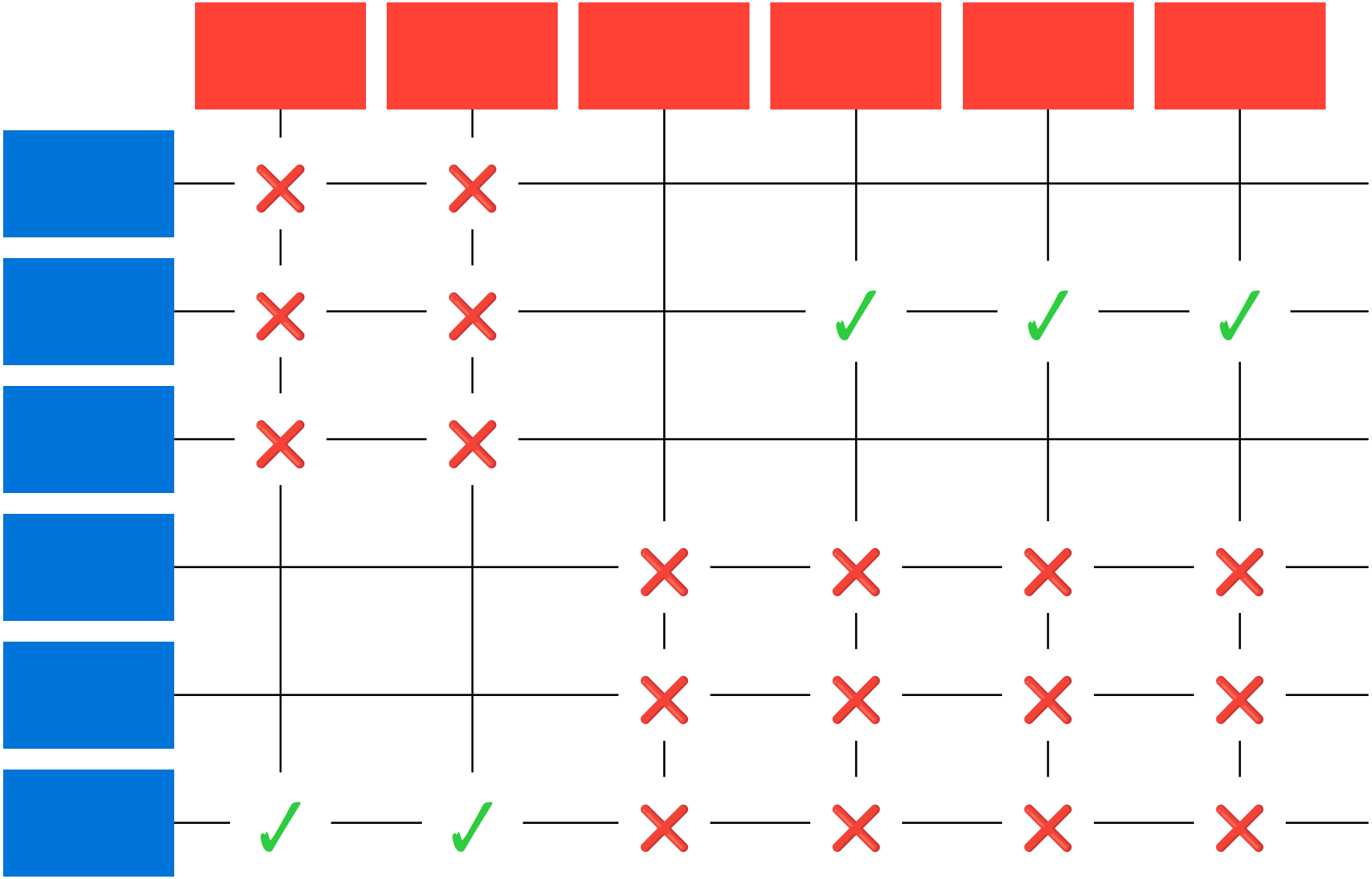
Join



Join



Join



Operator-at-a-Time


```
def join(key, In1, In2):
    output = []
    lookup = defaultmap([])

    for r in In1:
        lookup[r[key]] += r

    for s in In2:
        for r in lookup[s[key]]:
            output += [ r + s ]

    return output
```

Operator-at-a-Time

R_4	name	dest	P.t_no	T.route	stop
	Athena	48	Buffalo	Lake Shore Ltd.	Erie

```
r1 = product(Train, Stop)
r2 = filter(lambda row: ..., r1)
r3 = join('train_no', Passenger, r2)
r4 = filter(lambda row: ..., r3)
```

Operator-at-a-Time

R_4	name	dest	P.t_no	T.route	stop
	Athena	48	Buffalo	Lake Shore Ltd.	Erie

```
r1 = product(Train, Stop)
r2 = filter(lambda row: ..., r1)
r3 = join('train_no', Passenger, r2)
r4 = filter(lambda row: ..., r3)
Result = project([Train, Stop], r4)
```

Operator-at-a-Time

Result	name	dest
	Athena	48

```
r1 = product(Train, Stop)
r2 = filter(lambda row: ..., r1)
r3 = join('train_no', Passenger, r2)
r4 = filter(lambda row: ..., r3)
Result = project([Train, Stop], r4)
```

Operator-at-a-Time


```
def project(cols, In):
    output = []

    for r in In:
        out_row = []
        for c in cols:
            out_row += [ r[c] ]
        output += [ out_row ]

    return output
```

Why we don’t do operator-at-a-time

When is this a bad idea?


```
def filter(condition, In):
    output = []

    for r in In:
        if condition(r):
            output += [ r ]

    return output
```

The Memory Hierarchy

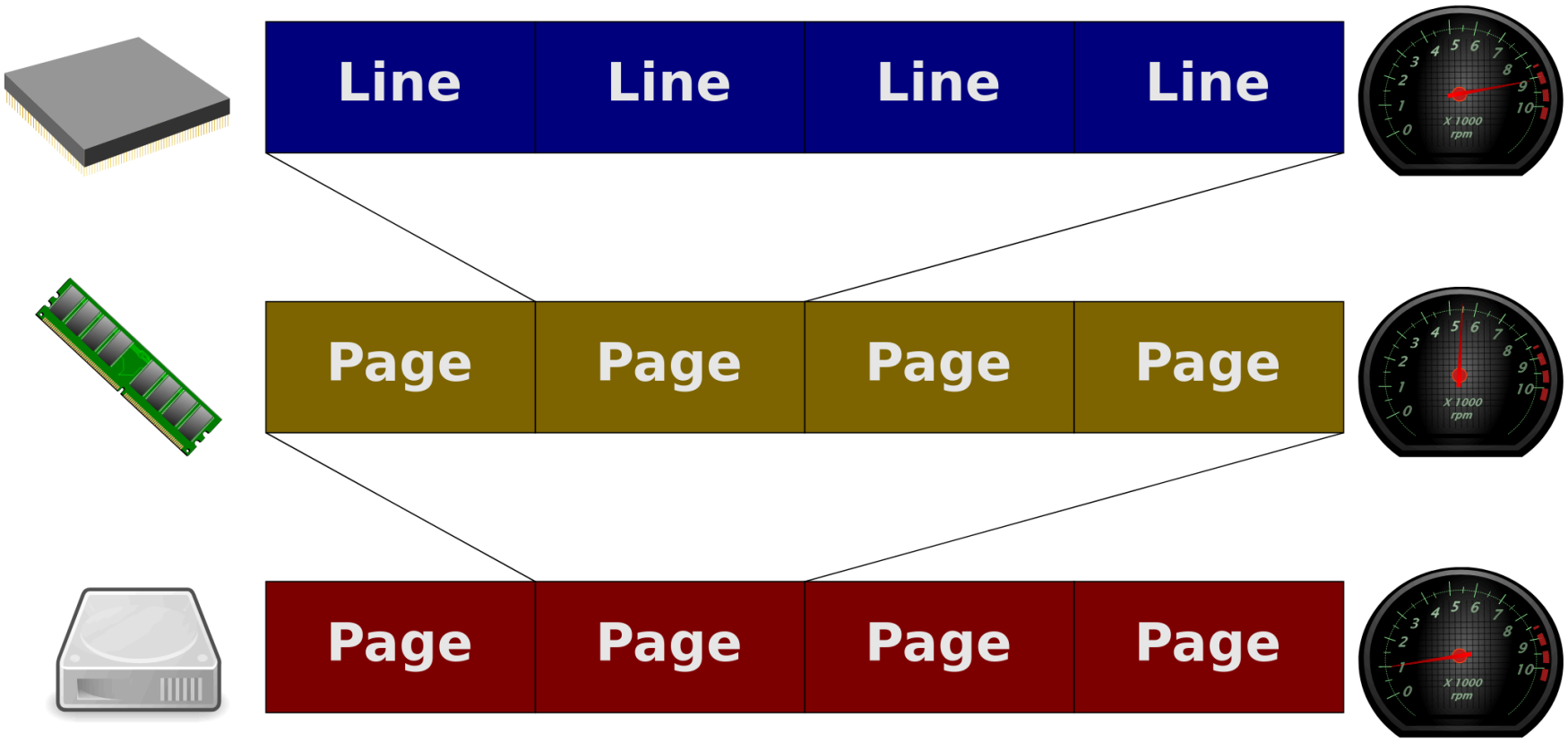


Image credit: Created with images from openclipart

```
def filter(condition, In):
    output = []

    for r in In:
        if condition(r):
            output += [ r ]

    return output
```

How do we measure the quality of an algorithm?

Algorithm Quality

- **Runtime Complexity**: How fast is the algorithm for a given input?
- **Memory Complexity**: How much memory do we need to run the algorithm on a given input?
- **IO Complexity**: How much data gets moved between layers of the hierarchy?

We can measure complexity...
- exactly ($3N$ pages of IO)
- asymptotically ($O(N)$ memory)

The model

- I: A limited "Internal" memory that starts off empty.
  - $|I|$ is the Memory Complexity
- E: An infinite "External" memory.

Actions

- $I[42] \leftarrow \mathbf{Compute}(I[0], I[1], ...)$: Adds to Runtime Complexity
- $I[12] \leftarrow \mathbf{Read}(E[974])$: Adds to IO Complexity
  - "Read Cost": Just reads
- $E[974] \leftarrow \mathbf{Write}(I[12])$: Adds to IO Complexity
  - "Write Cost": Just writes

The "IO Cost" or "IO Complexity" is Read cost + Write cost

1. **Read**($I[23]$)
2. **Read**($I[19]$)
3. **Read**($I[23]$)
4. **Read**($I[5]$)
5. **Read**($I[6]$)
6. **Read**($I[44]$)

Temporal and Spatial Locality

1. **Read**($I[23]$) ← Repeated Access
2. **Read**($I[19]$)
3. **Read**($I[23]$) ← Repeated Access
4. **Read**($I[5]$)
5. **Read**($I[6]$)
6. **Read**($I[44]$)

Temporal and Spatial Locality

1. **Read**($I[23]$) ← Repeated Access
2. **Read**($I[19]$)
3. **Read**($I[23]$) ← Repeated Access
4. **Read**($I[5]$)
5. **Read**($I[6]$)
6. **Read**($I[44]$)

Temporal Locality means repeated accesses to the *same* location, close in time.

Temporal and Spatial Locality

1. **Read**$(I[23])$ ← Repeated Access
2. **Read**$(I[19])$
3. **Read**$(I[23])$ ← Repeated Access
4. **Read**$(I[5])$ ← Almost Repeated Access
5. **Read**$(I[6])$ ← Almost Repeated Access
6. **Read**$(I[44])$

Temporal Locality means repeated accesses to the *same* location, close in time.

Temporal and Spatial Locality

1. **Read**($I[23]$) ← Repeated Access
2. **Read**($I[19]$)
3. **Read**($I[23]$) ← Repeated Access
4. **Read**($I[5]$) ← Almost Repeated Access
5. **Read**($I[6]$) ← Almost Repeated Access
6. **Read**($I[44]$)

Temporal Locality means repeated accesses to the *same* location, close in time.
Spatial Locality means repeated access to a *nearby* location, close in time
$I[a]$ and $I[b]$ are spatially local if $|a - b| < \varepsilon$ (for some small ε)

Temporal and Spatial Locality

1. **Read**$(I[23])$ ← Repeated Access
2. **Read**$(I[19])$
3. **Read**$(I[23])$ ← Repeated Access
4. **Read**$(I[5])$ ← Almost Repeated Access
5. **Read**$(I[6])$ ← Almost Repeated Access
6. **Read**$(I[44])$

Temporal Locality means repeated accesses to the *same* location, close in time.
Spatial Locality means repeated access to a *nearby* location, close in time
$I[a]$ and $I[b]$ are spatially local if $|a - b| < \varepsilon$ (for some small ε)

Using Temporal and Spatial Locality

1. **Read**($I[23]$)
2. **Read**($I[19]$)
3. **Read**($I[23]$)
4. **Read**($I[5]$)
5. **Read**($I[6]$)
6. **Read**($I[44]$)

Using Temporal and Spatial Locality

1. **Read**($I[23]$)
2. **Read**($I[19]$)
3. **Read**($I[23]$) ← We can skip temporally local reads by caching values.
4. **Read**($I[5]$)
5. **Read**($I[6]$)
6. **Read**($I[44]$)

Using Temporal and Spatial Locality

1. **Read**($I[23]$)
2. **Read**($I[19]$)
3. **Read**($I[23]$) $\leftarrow$ We can skip temporally local reads by caching values.
4. **Read**($I[5]$)
5. **Read**($I[6]$) $\leftarrow$ Spatially local reads may be temporally local at a lower level.
6. **Read**($I[44]$)

Using Temporal and Spatial Locality

1. **Read**($I[23]$)
2. **Read**($I[19]$)
3. **Read**($I[23]$) $\leftarrow$ We can skip temporally local reads by caching values.
4. **Read**($I[5]$)
5. **Read**($I[6]$) $\leftarrow$ Spatially local reads may be temporally local at a lower level.
6. **Read**($I[44]$)

We want to maximize spatial and temporal locality!

Using Temporal and Spatial Locality

1. **Read**($I[23]$)
2. **Read**($I[19]$)
3. **Read**($I[23]$) ← We can skip temporally local reads by caching values.
4. **Read**($I[5]$)
5. **Read**($I[6]$) ← Spatially local reads may be temporally local at a lower level.
6. **Read**($I[44]$)

We want to maximize spatial and temporal locality![1]

[1] Ram is big now. Modern SSDs are fast. Your mileage may vary.

Using Temporal and Spatial Locality

1. **Read**($I[23]$)
2. **Read**($I[19]$)
3. **Read**($I[23]$) ← We can skip temporally local reads by caching values.
4. **Read**($I[5]$)
5. **Read**($I[6]$) ← Spatially local reads may be temporally local at a lower level.
6. **Read**($I[44]$)

We want to maximize spatial and temporal locality![2]

[2] Ram is big now[3]. Modern SSDs are fast. Your mileage may vary.

[3] Although Sam Altman doesn't want you to have any.

Every layer is the same

Measuring IO Cost

- **Cache Lines** : ~16-128B for Cache
- **Pages** : ~4KB for RAM↔Disk
- **Blocks** : ~64MB for HDFS

For simplicity, we'll measure memory in either "Records", or "Pages"

```
def filter(condition, In):
    output = []

    for r in In:
        if condition(r):
            output += [ r ]

    return output
```

1. If $|\text{In}|$ is N records:
   - What is the asymptotic memory complexity of this algorithm?
   - What is the exact IO complexity of this algorithm (in records)?

Assumptions...

- Does `output` need to be written back to disk?
  - Does `output` fit into memory?
- Does `In` arrive in memory (does it fit?)

How do the answers to these questions change the complexities?

Mem / IO Complexity

```
def filter(condition, In):
    buffer = []
    out_file = File()

    for r_page in In.by_page:
        for r in r_page:
            if condition(r):
                buffer += [ r ]
            if len(output) > BUFFER_LEN:
                out_file.write(buffer)
                buffer = []
    out_file.write(buffer)

    return out_file
```

How does this change the complexities? ($|\mathrm{In}|$ is still N records)

```
def product(In1, In2):
    output = []

    for r in In1:
        for s in In2:
            output += [ r + s ]

    return output
```

1. How do we rewrite this to use disk?
2. If $|\text{In1}|$ and $|\text{In2}|$ are N records:
   - What is the asymptotic memory complexity of this algorithm?
   - What is the exact IO complexity of this algorithm (in records)?

```
def join(key, In1, In2):
    output = []
    lookup = defaultmap([])
    for r in In1:
        lookup[r[key]] += r
    for s in In2:
        for r in lookup[s[key]]:
            output += [ r + s ]
    return output
```

1. How do we rewrite this to use disk?
2. If $|\mathrm{In1}|$ and $|\mathrm{In2}|$ are N records:
   - What is the asymptotic memory complexity of this algorithm?
   - What is the exact IO complexity of this algorithm (in records)?

We sure are reading and writing a lot!

How do we do better?

We can implement operators as iterators.

```
class Iterator:
    def next(self) -> Option[Record]:
        pass
```

Operator iterators compose and the final result is read off the final iterator:

```
def print_result(query_iterator):
    while (row := query_iterator.next()) is not None
        print(row)
```

This is called the 'Volcano' model

Table Scan


```
class TableIterator:
    def __new__(self, file):
        self.file = file
        self.buffer = []

    def next(self) -> Option[Record]:
        if len(self.buffer) == 0
            self.buffer = self.file.read_page()
        return self.buffer.pop(0)
```

Table Scan


```
class TableIterator:
    def __new__(self, file):
        self.file = file
        self.buffer = []

    def next(self) -> Option[Record]:
        if len(self.buffer) == 0
            self.buffer = self.file.read_page()
        return self.buffer.pop(0)
```

What is the...

- Memory complexity?

Table Scan


```
class TableIterator:
    def __new__(self, file):
        self.file = file
        self.buffer = []

    def next(self) -> Option[Record]:
        if len(self.buffer) == 0
            self.buffer = self.file.read_page()
        return self.buffer.pop(0)
```

What is the...

- Memory complexity?
- IO Complexity?

Project Iterator


```
class ProjectIterator:
    def __new__(self, columns, in):
        self.columns = columns
        self.in = in

    def next(self) -> Option[Record]
        ???
```

Project Iterator


```
class ProjectIterator:
    def __new__(self, columns, in):
        self.columns = columns
        self.in = in

    def next(self) -> Option[Record]
        row = self.in.next()
        row = [ row[c] for c in self.columns ]
        return row
```

Project Iterator


```
class ProjectIterator:
    def __new__(self, columns, in):
        self.columns = columns
        self.in = in

    def next(self) -> Option[Record]
        row = self.in.next()
        row = [ row[c] for c in self.columns ]
        return row
```

What is the...

- Memory complexity?
- IO Complexity?

Filter Iterator


```
class FilterIterator:
    def __new__(self, condition, in):
        self.condition = condition
        self.in = in

    def next(self) -> Option[Record]
        ???
```

Filter Iterator


```
class FilterIterator:
    def __new__(self, condition, in):
        self.condition = condition
        self.in = in

    def next(self) -> Option[Record]
        do:
            row = self.in.next()
        while row is not None and not condition(row):
        return row
```

Filter Iterator


```
class FilterIterator:
    def __new__(self, condition, in):
        self.condition = condition
        self.in = in

    def next(self) -> Option[Record]
        do:
            row = self.in.next()
        while row is not None and not condition(row):
        return row
```

What is the...

- Memory complexity?
- IO Complexity?

Union Iterator


```
class UnionIterator:
    def __new__(self, in1, in2):
        self.in1 = in1
        self.in2 = in2

    def next(self) -> Option[Record]
        ???
```

Union Iterator


```
class UnionIterator:
    def __new__(self, in1, in2):
        self.in1 = in1
        self.in2 = in2

    def next(self) -> Option[Record]
        if (row := self.in1.next()) is not None:
            return row
        else:
            return self.in2.next()
```

Union Iterator


```
class UnionIterator:
    def __new__(self, in1, in2):
        self.in1 = in1
        self.in2 = in2

    def next(self) -> Option[Record]
        if (row := self.in1.next()) is not None:
            return row
        else:
            return self.in2.next()
```

What is the...

- Memory complexity?
- IO Complexity?

Course TA

Pratik Pokharel (More info soon)

Deliverables

- AI Quiz Due Tomorrow!
- Checkpoint 0 PAST DUE!
  - Reach out to me if you haven't finished it yet!