Rensselaer

why not change the world?®

Measuring Repeatable Read with Read/Write Locking Performance

ECSE 6320 | Rojan Karn | Almog Cohen | 4/24/2023

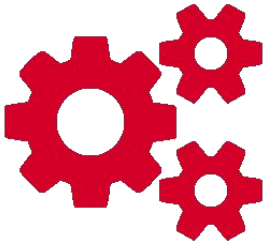
Agenda

1. Motivation
2. Transaction Isolation
3. Database Implementation
4. Repeatable Read Implementation
5. Results and Findings

Motivation

Problem

- Data inconsistency in database systems with concurrent transactions
  - Multiple transactions trying to update the same data at the same time
  - One transaction trying to read data that has been updated by another transaction
- Need to ensure isolation between transactions using concurrency control techniques



Solution

- Try to emulate the effect of running transactions in parallel using repeatable read
- Implement read/write locking to ensure data integrity

Motivation

Database	
ID	Balance
0	10
1	10

- If run sequentially, Transaction 2 will always print $20
- What happens if Transaction 1 finishes in the middle of Transaction 2?

Thread 1 - Transfers $10 from ID 0 to 1

```
begin_tx
a = readId(0, “Balance”)
b = readId(1, “Balance”)
writeId(0, a - 10)
writeId(1, b + 10)
commit_tx
```

Thread 2 - Adds Balance of ID 0 and 1

```
begin_tx
a = readId(0, “Balance”)
b = readId(1, “Balance”)
print(a + b)
commit_tx
```

Motivation

Thread 1 - Transfers $10 from ID 0 to 1

```
begin_tx
a = readId(0, "Balance")
b = readId(1, "Balance")
writeId(0, a - 10)
writeId(1, b + 10)
commit_tx
```

Thread 2 - Adds Balance of ID 0 and 1

```
begin_tx
a = readId(0, "Balance")
b = readId(1, "Balance")
print(a + b)
commit_tx
```

	Thread 1	Thread 2
Timestep 1	`commit_tx`	`a = readId(0, "Balance")` → $10
Timestep 2		`b = readId(1, "Balance")` → $20
Timestep 3		`print(a + b)` → $30 Problem!

Transaction Isolation Levels

Least Desirable

Most Desirable

Isolation Level	Description
Read uncommitted	Allows seeing uncommitted data changes from other transactions
Read Committed	Allows reading only committed data, but non-repeatable reads are possible
Repeatable Read	Allows reading the same data repeatedly in the same transaction
Serializable	Provides the highest level of isolation, guarantees correctness but with a performance penalty

Custom transaction/query language

```
begin_tx
a = readId(0, “Balance”)
b = readVal(“Name”, “George”, “Balance”)
if (a >= 10)
   writeId(0, a - 10)
   writeVal(“Name”, “George”, b + 10)
else
   abort_tx
endif
commit_tx
```

Custom transaction/query language

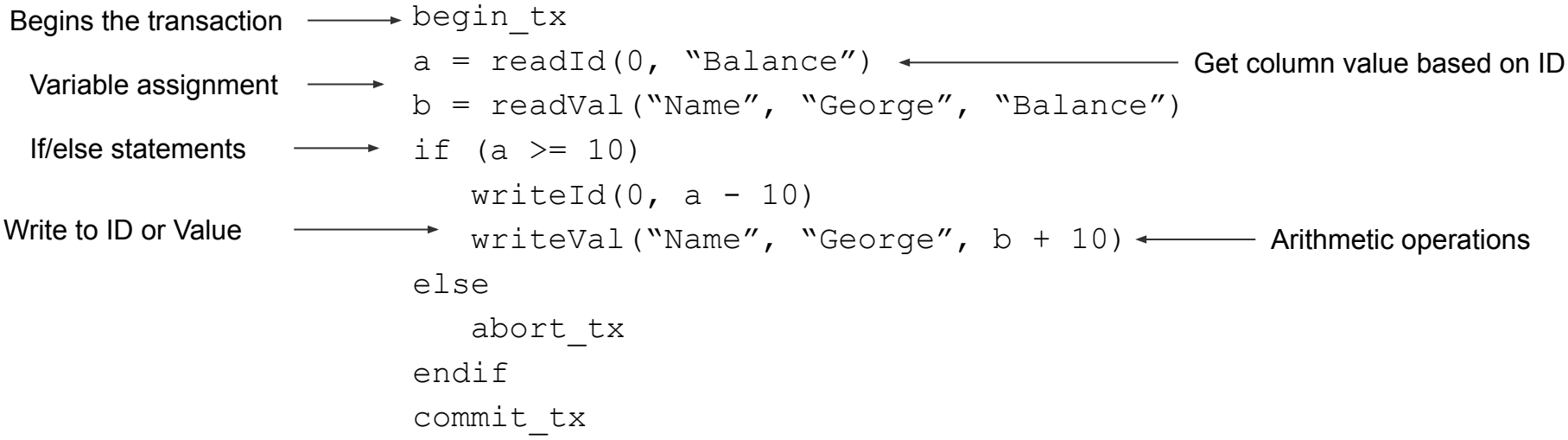
Begins the transaction →

Variable assignment →

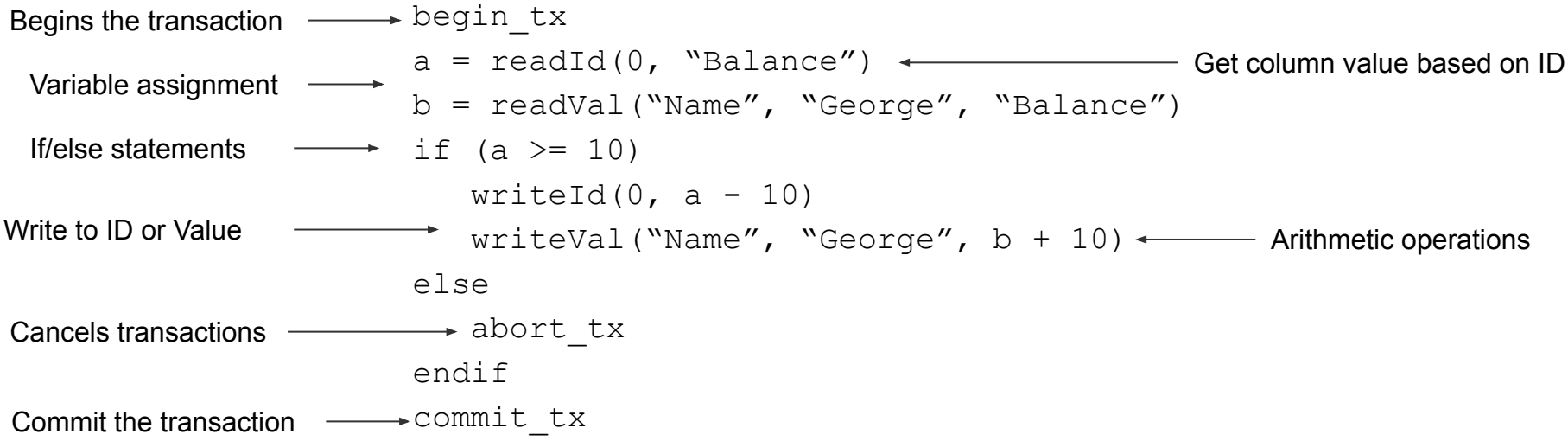
```
begin_tx
a = readId(0, "Balance")
b = readVal("Name", "George", "Balance")
if (a >= 10)
   writeId(0, a - 10)
   writeVal("Name", "George", b + 10)
else
   abort_tx
endif
commit_tx
```

← Get column value based on ID

Custom transaction/query language



Custom transaction/query language



Database Implementation

- Current implementation stores data in void* hash table of incrementing unique IDs
  - O(1) for data retrieval with O(n) for search
  - Large overhead to store data on disk compared to B-Tree
- Database defined based on custom schema
- Many robust database systems use B-Trees
  - Fast storage and retrieval of large amounts of data
  - Highly scalable and can easily adapt to increasing amounts of data
  - Minimize disk access by storing data in nodes that are optimized for disk blocks
  - Self-balancing as data is inserted

Database Implementation

- Table design with unique IDs at every row
- Define table schemas with number of columns and their type

```
DatabaseTableSchema schema = {
    .columnCount = 2,
    .columnNames = new string[2]{"Name", "Balance"},
    .columnTypes = new int[2]{DATABASE_TYPE_STRING, DATABASE_TYPE_INT}
};
```

- Supported functions
  - `setID(ID, ColumnName, Value)` → Set a value in particular row, column
  - `setVal(SearchColumn, SearchValue, ColumnName, Value)` → Set a value in particular row, column by search
  - `getID(ID, ColumnName)` → Get a value in particular row, column
  - `getVal(SearchColumn, SearchValue, ColumnName)` → Get a value in particular row, column by search

Repeatable Read

- Repeatable read: isolation level in database systems
  - Ensures transactions see the same data throughout the entire transaction
  - Other transactions modifying this data at the same time will not affect the transaction
- Used in applications that require a high degree of data consistency
- Improves performance by reducing number of conflicts between transactions
- Implemented with **locking**

SHARED lock

- Transactions can read the same data simultaneously
- Prevents modifying the data
- Used when transaction only needs to *read* data

EXCLUSIVE lock

- Prevents other transactions from reading or writing the locked data
- Used when a transaction needs to write/modify data
- Other transactions cannot lock this data

Two-Phase Locking (2PL)

Growing phase	Locks are *acquired*, no locks are released
Shrinking phase	Locks are *released*, no locks are acquired

- Transaction must obtain all necessary locks before executing transactions
  - Must not release any locks until execution is completed
- Prevents conflicting transactions from overlapping in their execution
  - One transaction must wait for the other to finish before starting

Importance of Locking

- **Data consistency**
  - Transactions will always see the same data, even if other transactions are updating the data
- **Deadlock prevention**
  - Occurs when two or more transactions are waiting for each other to release a lock
  - Locking ensures that transactions acquire locks in a specific order
- **Lost update prevention**
  - Occurs when a transaction updates data that has been updated by another transaction
  - Locking ensures transactions only update data that has not been updated by another transaction

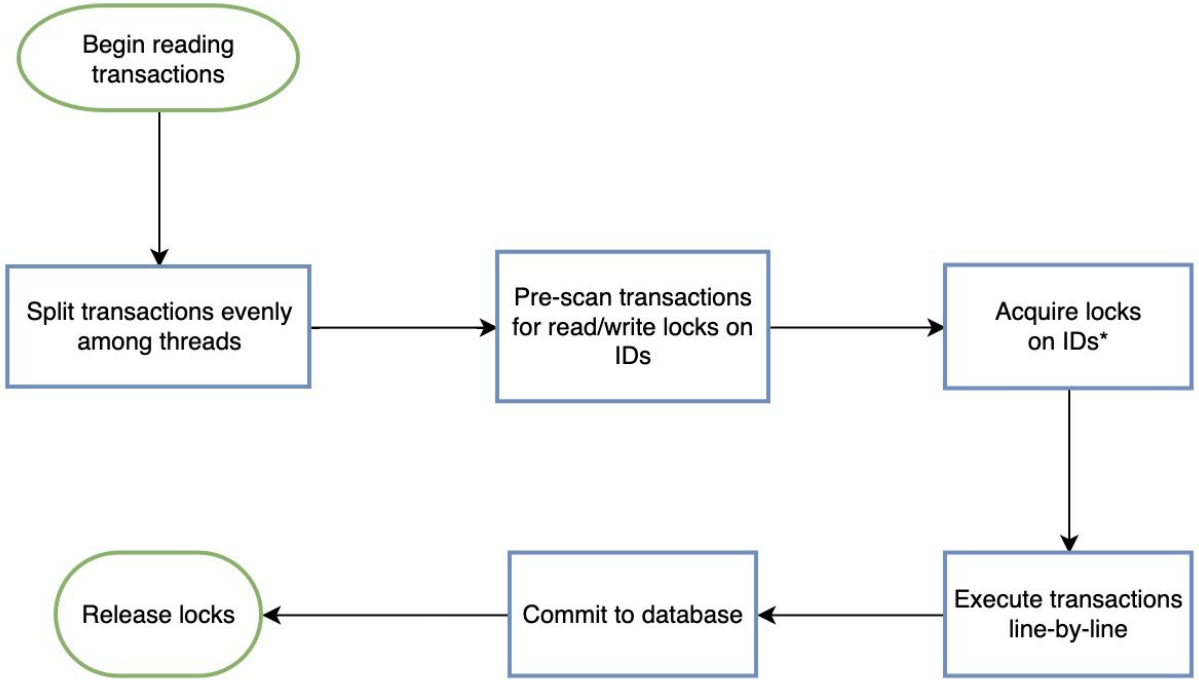
Transaction Manager: ReadWriteLockingTable

ReadWriteLockingTable (unordered_map)

Data ID	Read Locked?	Write Locked?	
0	No	Yes	⟸ *SHARED lock*
57	No	No	⟸ *Unlocked*
421	Yes	Yes	⟸ *EXCLUSIVE lock*
…	…	…	

- **SHARED lock**: Write Locked = Yes
  - Used for read-only IDs
- **EXCLUSIVE lock**: Read Locked = Yes && Write Locked = Yes
  - Used for IDs that need to be modified

Locking Data for Transactions



Transaction Lifecycle

Growing phase:

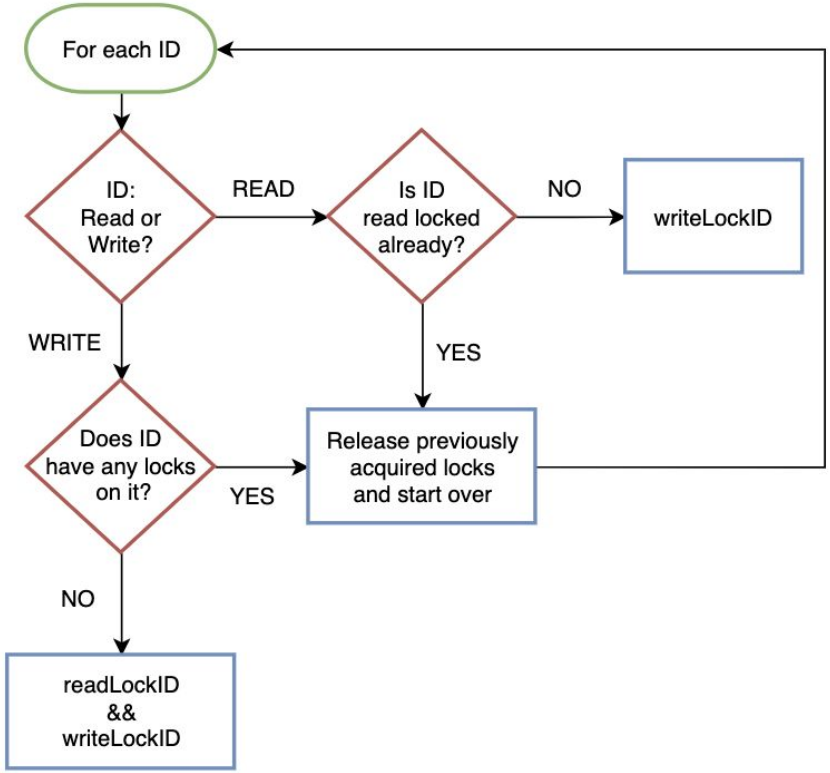
- Acquire all read/write locks before beginning execution

Shrinking phase:

- Release all locks after transaction has been committed

Acquiring Locks

- List of IDs that need to be locked generated by transaction pre-scan
- If conflict between transactions, wait for all necessary IDs to be unlocked
  - Function returns when all IDs have been successfully locked
- Avoids deadlocking
  - Tries to acquire all locks at once
  - Does not hold any locks indefinitely



Repeatable Read: Other Implementations

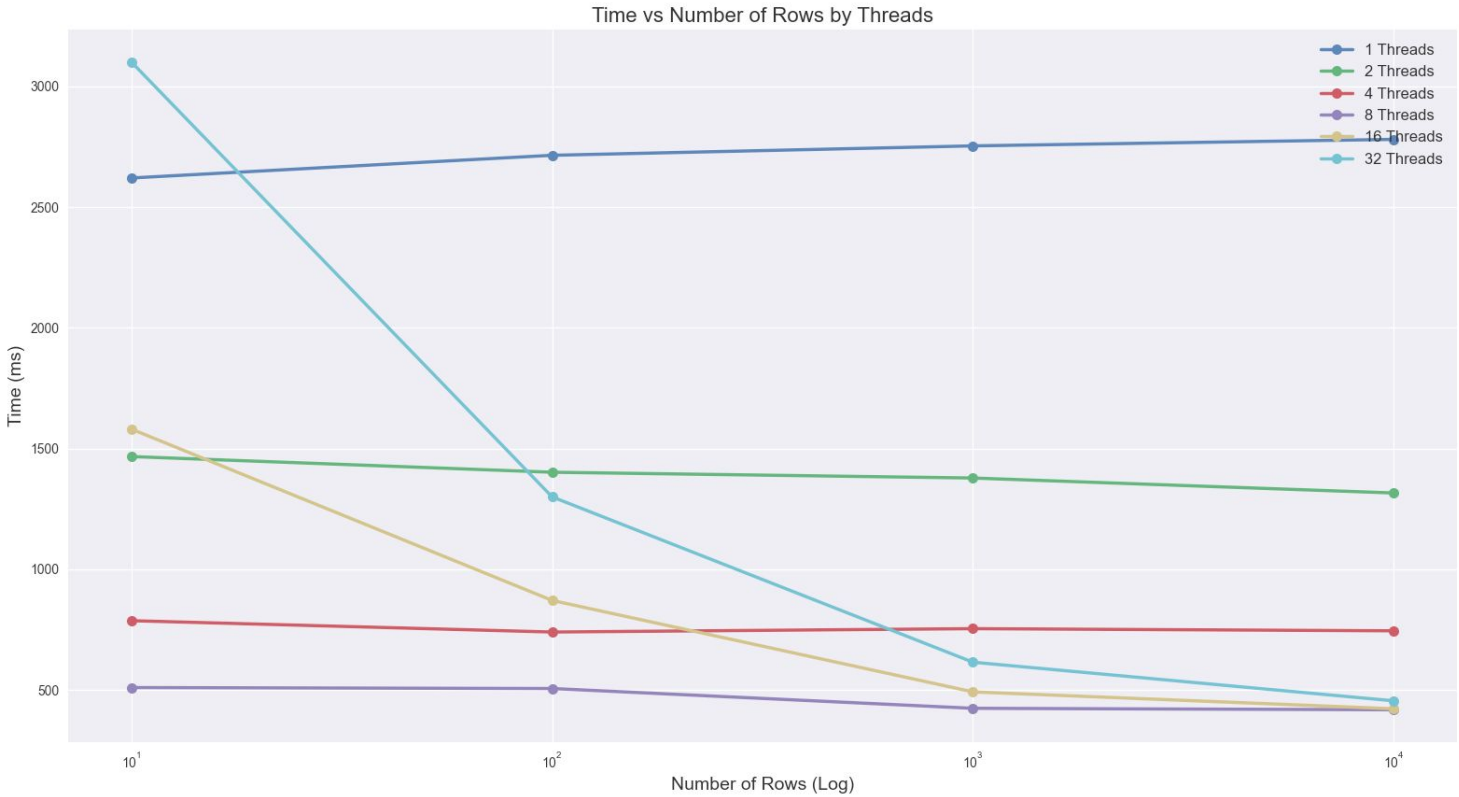
Read/Write Locking	Snapshot Isolation	Optimistic Concurrency Control
• Acquire locks on data that transaction needs to access • When transaction is complete, release locks • High performance	• Transactions see data from a snapshot taken at the start of the transaction • Modifications to database made after snapshot are not visible to transaction • High storage and computational overhead	• Assume transactions will not conflict with each other • Uses timestamps on data to check for conflicts • If conflict, roll back transaction • Used in environments with low data contention

SIMD

- SIMD used for database searching
- Database size padded to be divisible by 8
- Utilizes AVX2 comparator operator

Loads value to search for →	12171	12171	12171	12171	12171	12171	12171	12171
Loads 8 search values at a time →	4613	53130	3230	33131	97100	12171	32304	99200
Generates mask →	0	0	0	0	0	1	0	0

Performance Results



Tested on 100,000 random access read/write transactions

Performance Results

- Databases with low cardinality or high simultaneous ID modifications should remain sequentially accessed
- Repeatable Read with Read/Write Locking best suited for high cardinality databases with lots of random access
- As more threads are added, probability of collision increases which diminishes the benefit → Scale number of threads based on database demands

Future Work

- Develop more efficient locking system
  - Use queue for pending lock requests from concurrent transactions
- Implement logging in database system
  - Ensure atomicity: undo, redo
- Crash recovery
  - Add checkpoint entries to log
  - Use write-ahead logging to restore database after crash

Questions?

PROPER AND IMPROPER USE OF A POWERPOINT

- You should:
  - Use the template
  - Use font “Arial”
  - Use good-quality images
  - Make punctuation and capitalization consistent
  - Use bullets
  - Include key points and concepts
  - Use diagrams to illustrate complex concepts

- You should not:
  - Use PowerPoint as a handout, putting all spoken words on slides
  - Read your slides or speak to them
  - Use whole sentences or paragraphs on your slides (except for quotes)
  - Use complicated graphics or charts
  - Use flashy transitions and animations
  - Use poor-quality or web-napped images

BOXED CONTENT SLIDE

Box 1 Header

- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…

Box 2 Header

- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…

Box 3 Header

- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…

BOXED CONTENT SLIDE 2

Box 1 Header

- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…

Box 2 Header

- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…
- Bullet text here…

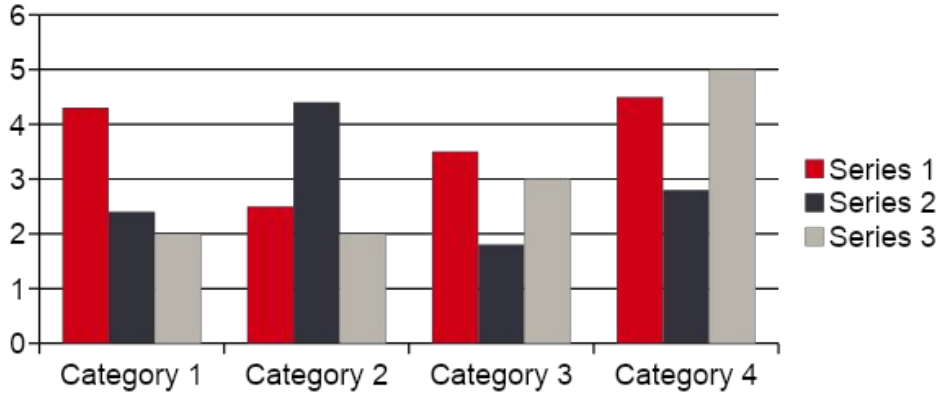
SLIDE LAYOUT OPTIONS

- Title slide
  - Used for presentation title only
- Closing slide
  - No content to be added
- Content w/ bullets
  - Style to be used for text (bullets, etc.)
- Photo or chart slides
  - Layout using large image or one to the left or right with the footer
- Divider slide (optional)
  - Use to separate topics or speakers

- Small collection of Rensselaer images are placed throughout the deck
  - Backgrounds for section headers
  - Backgrounds for quotes
  - Any high-quality image or chart can be used
  - Do not use as a background for bulleted lists

Including charts and graphs

- In the insert tab, you can create your own charts in PPT
- Multiple options of charts
- Charts and graphs can help the viewer easily understand complex information

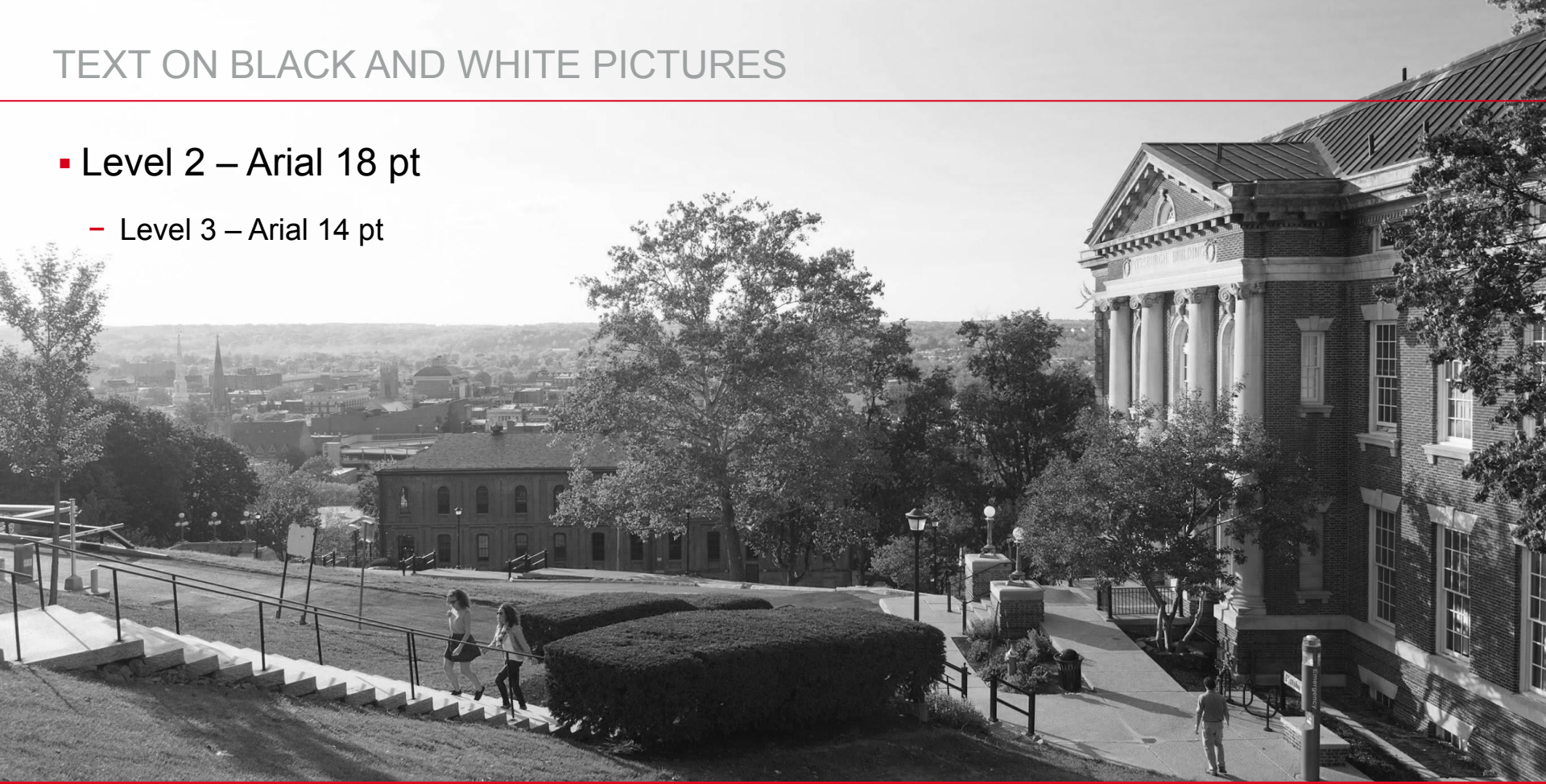


TEXT ON COLOR PICTURES

- Level 2 – Arial 18 pt
  - Level 3 – Arial 14 pt

TEXT ON BLACK AND WHITE PICTURES

- Level 2 – Arial 18 pt
  - Level 3 – Arial 14 pt



To get more information on brand style, templates and PowerPoint guidelines reach out to scer.rpi.edu

- Level 2 – Arial 18 pt
  - Level 3 – Arial 14 pt


Rensselaer
Class of
1992

Rensselaer

why not change the world?®

Rensselaer

why not change the world?®