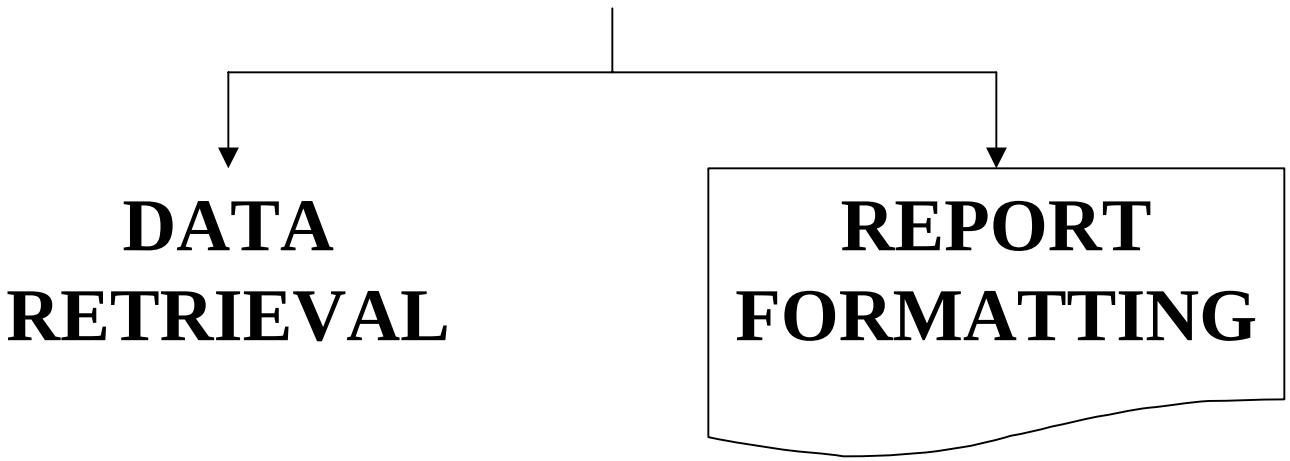
MICROBANK

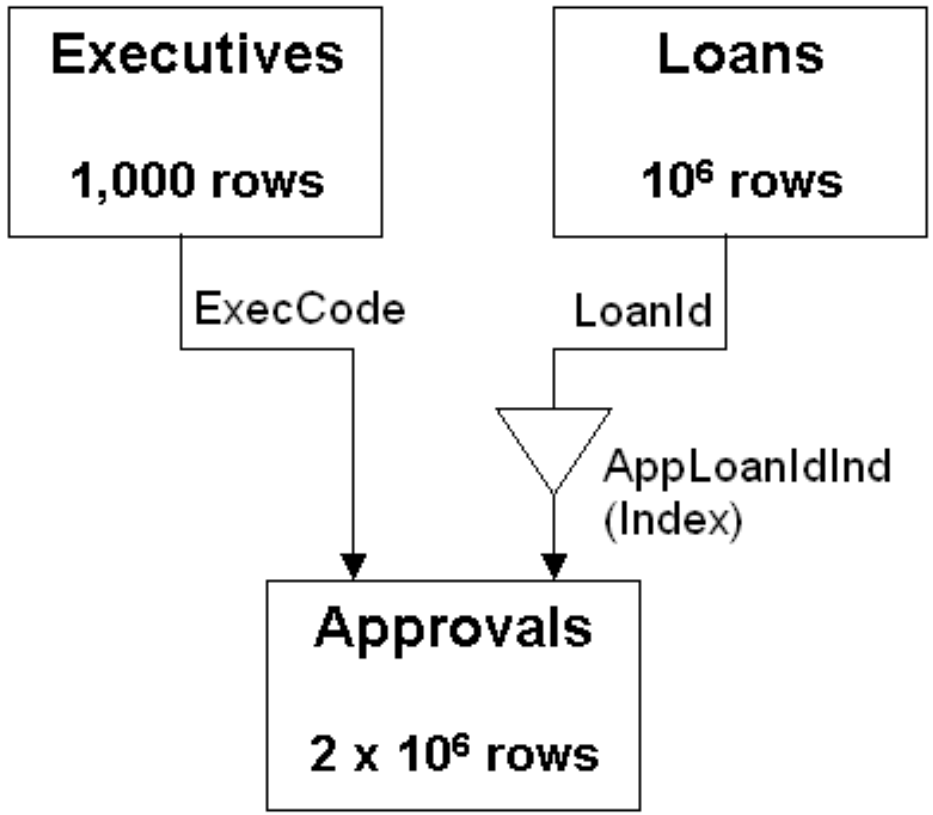
What Was Needed

REPORT

- Loan totals for individual executive
- Grand total of all loans issued
- Reporting time cannot exceed 2 hours!



Database Tables



Database Access Times

- Unsorted row fetch from cursor
  - 0.8 ms
  - declare loanCur cursor for select loanId from loans
  - fetch loanCursor into loanId
- Select operation using indexes
  - 1.5 ms
  - select * from approvals
- Select operation not using indexes
  - N x 0.5 ms, where N = number of rows in the table
  - select * from executives

Rule #1

K.I.S.S.

Keep It Simple, Stoopid

A Simplistic Solution

```
select count(*), sum(loanAmount)
from   loans, approvals
where  appExecCode = execCode and
       loanId = appLoanId and
       loanStatus = 0
```

- It counts the number of non-performing loans and the values associated to a particular executive
- Executes in α time
- Run twice per executive
- 1,000 executives in the report
- **2,000 α to generate the report**

What The Database Does

```
1. select * from loans
   where  loanStatus = 0
2. For each selected loans record
   a. select * from approvals
      where  appLoanId = loanId and
             appExecCode = execCode
   b. For each selected approvals record
      i. Increment count and sum value
```

Breaking Down α

OPERATION	TIME (hours)
Un-indexed select from loans table (1)	0.14
Indexed select from the approvals table (2.a)	0.42
TOTAL	**0.56**

Time Cost

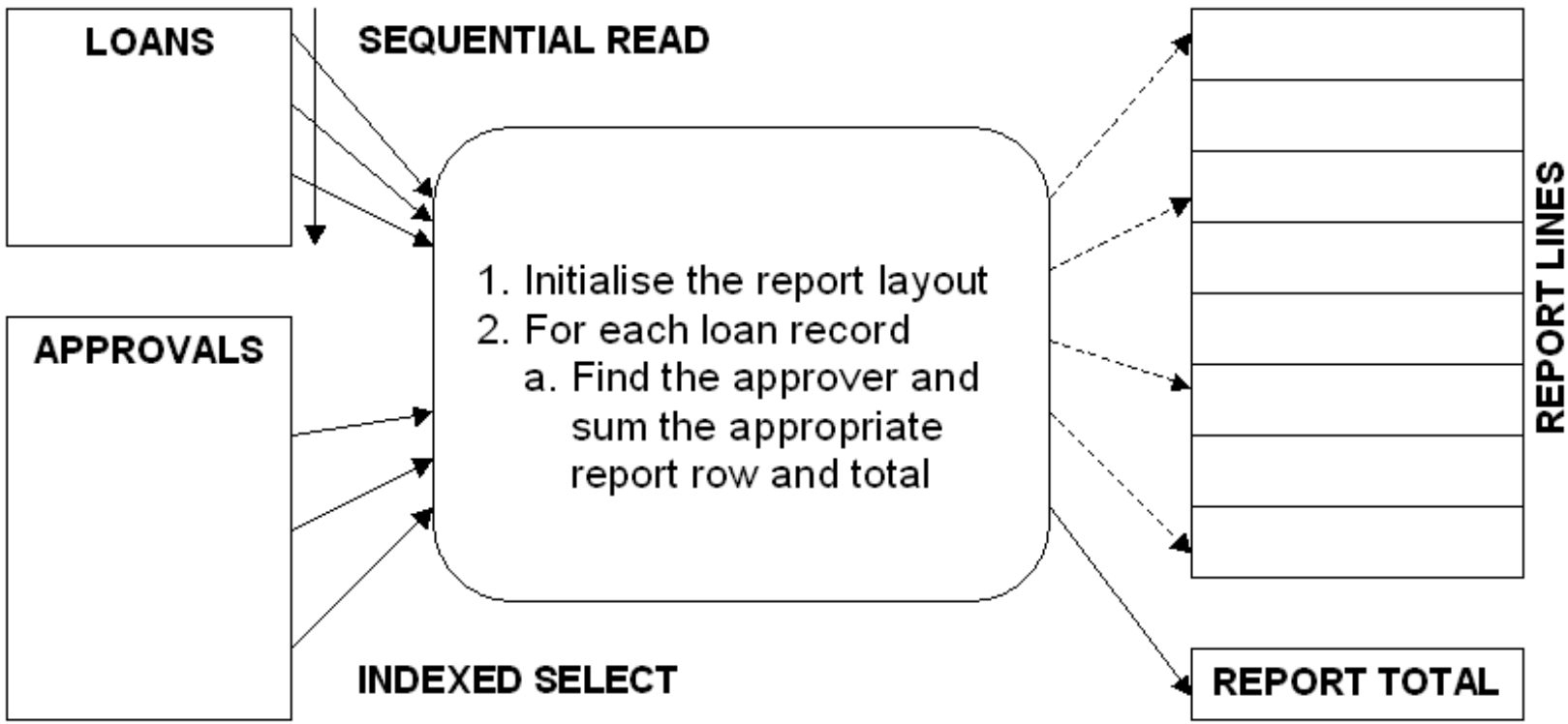
Procedure	Time (Hours)
2,000 α	1,120
Un-indexed select of loans to count totals	0.14
Un-indexed select of loans to count non-performing totals	0.14
TOTAL	**1,120.28**

Rule #2

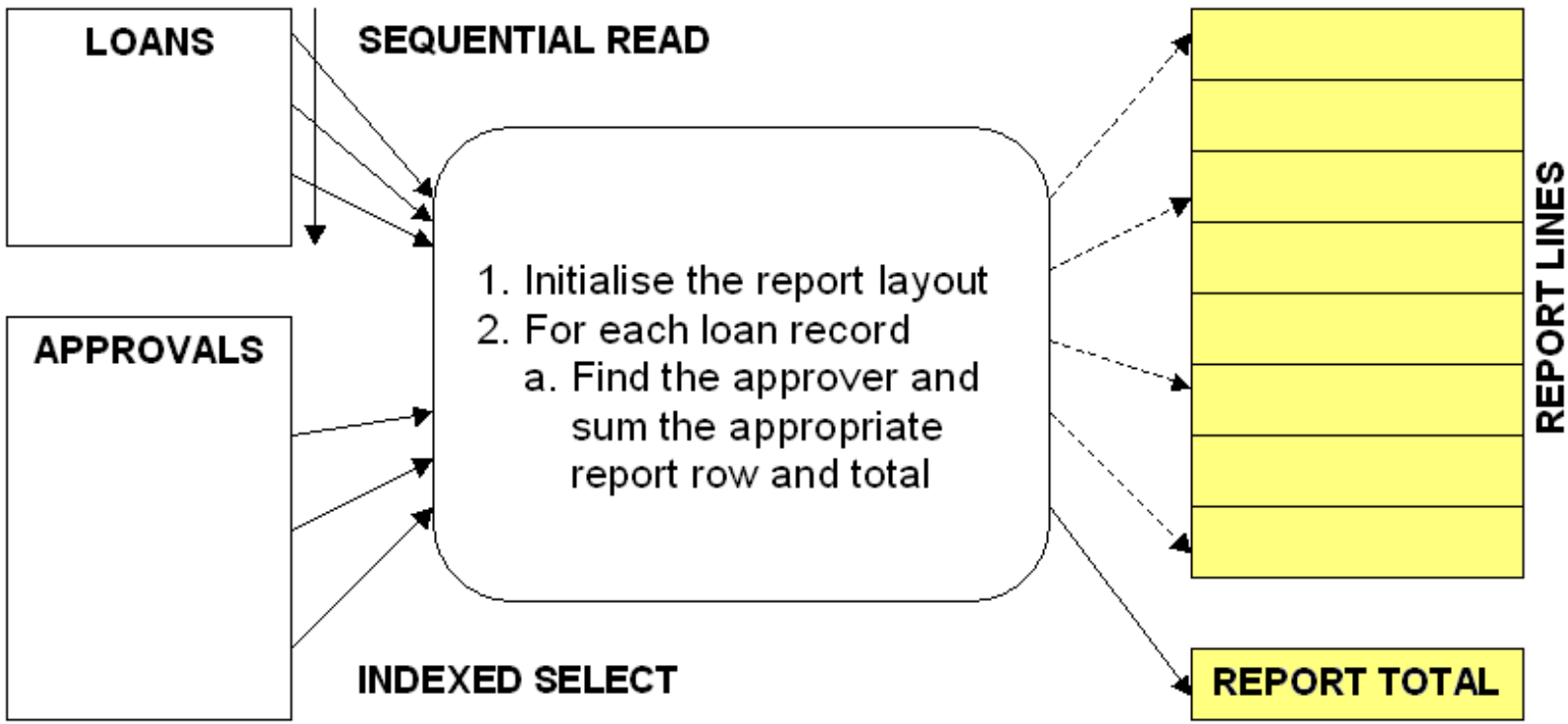
For every problem, there is a solution that is simple...

BUT SOMETIMES WRONG!

Push Algorithm Overview



Push Algorithm Overview



Representing The Report

```
struct reportRow {
   int      execCode;      // Executive Code
   char     execName[21]; // Executive Name
   int      loanCount;     // Number of loans issued
   double   loanValue;     // Value of loans issued
   int      nonPerfCount; // Number of non-
                          // performing loans issued
   double   nonPerfValue; // Value of non-
                          // performing loans issued
};
```

- Each instance of reportRow represents a row in the report
- Another reportRow instance can be used to represent the total

Memory Overhead

Per Record			
Data type	Size (bytes)	Required	Total (bytes)
Integer	4	3	12
Double	8	2	16
Character	1	21	21
Alignment padding	1	3	3
TOTAL			**52**

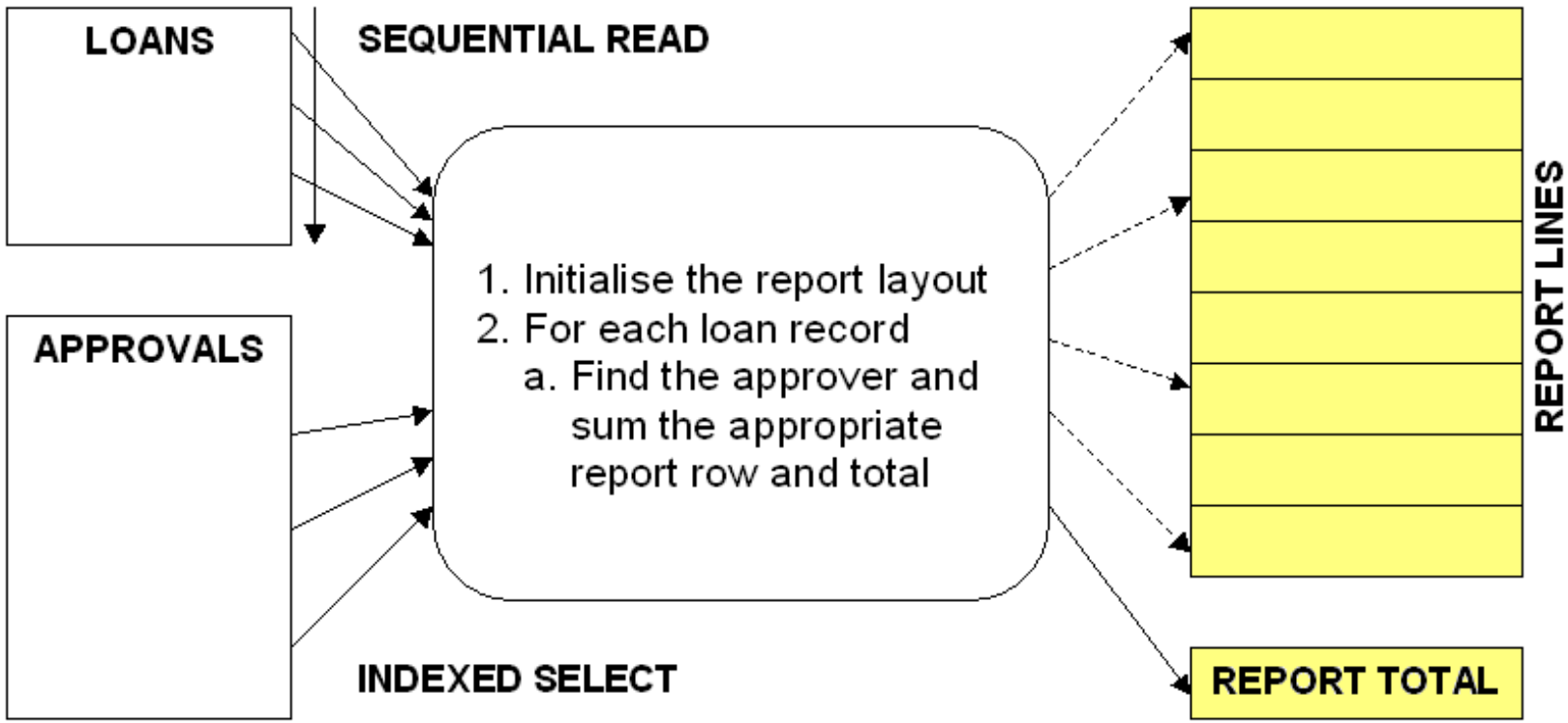
- For 1,000 executives, memory overhead = **52 kilobytes**

Build The Report Structure (α)

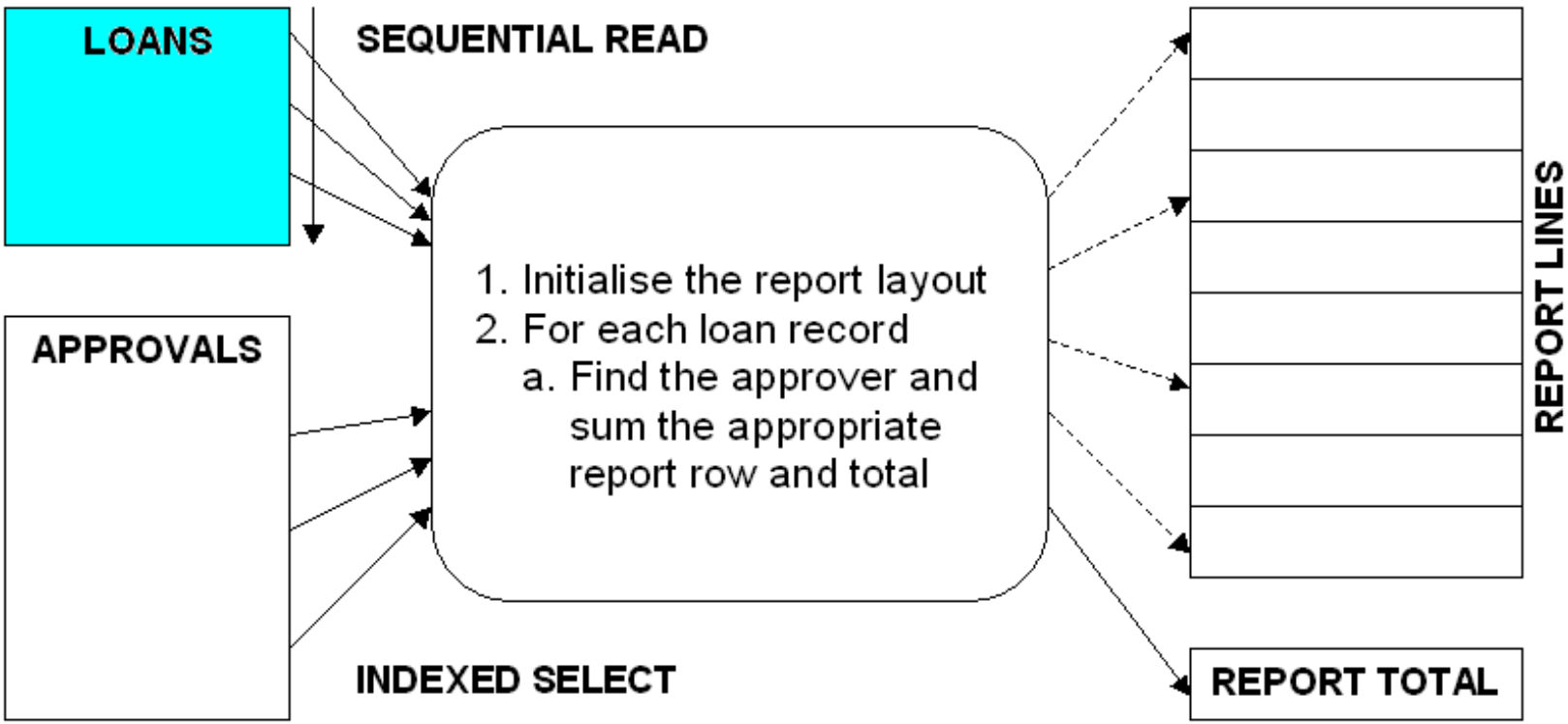
```
1. select execCode, execName
   from   executives
2. For each executive record selected
   a. Initialise a reportRow for the
      executive
3. Initialise a reportRow for the total
```

- Time taken = 0.8 ms x 1,000 = 0.8 s

Push Algorithm Overview



Push Algorithm Overview

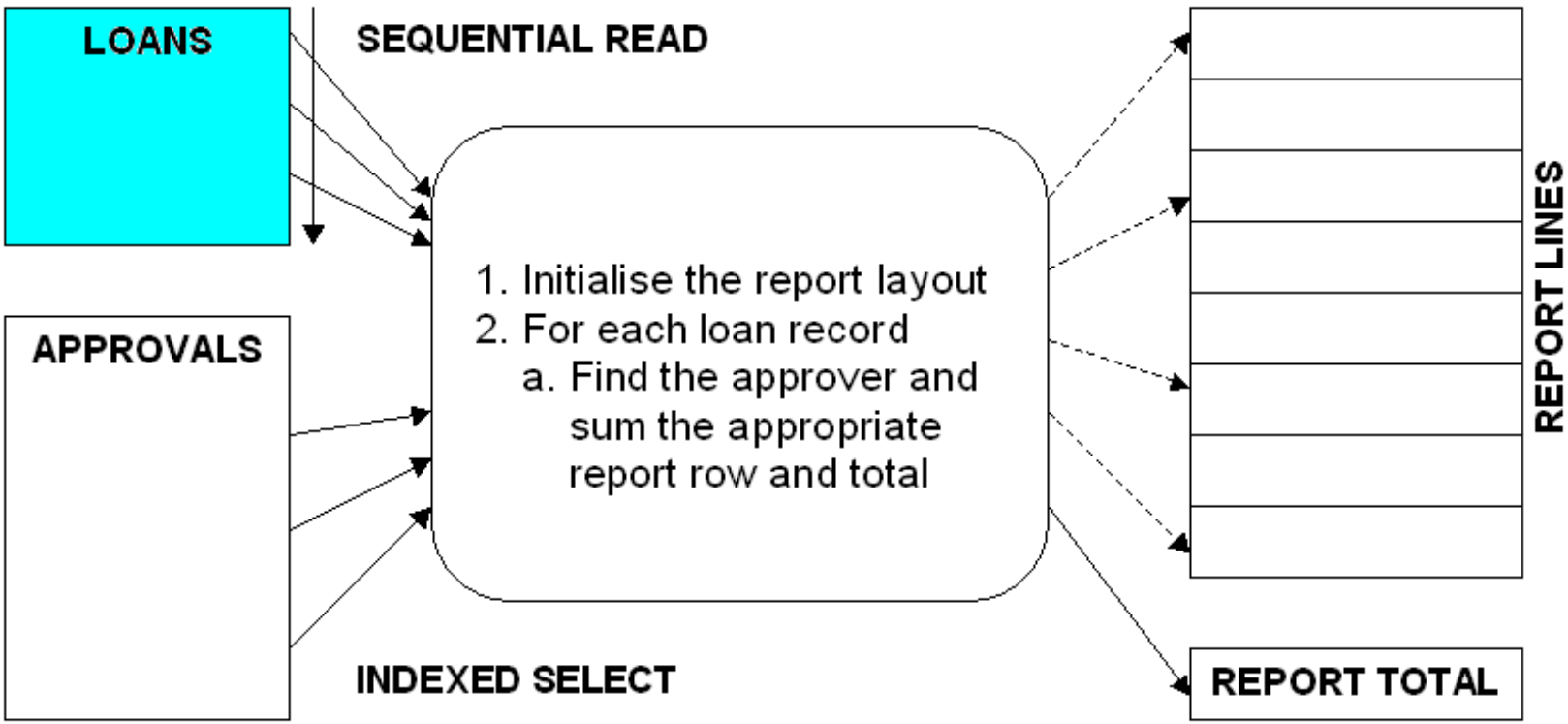


Gather Loan Details (β)

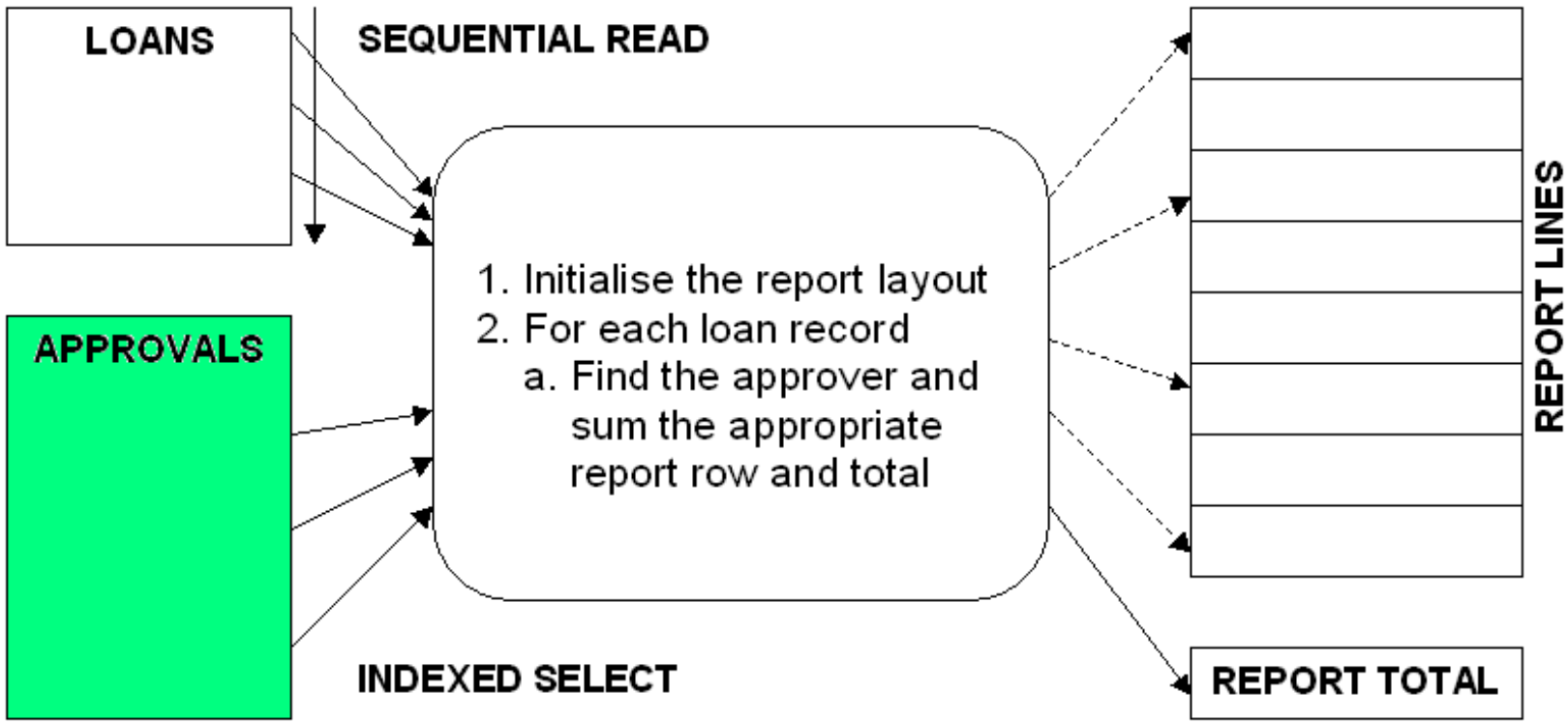
1. `select loanId, loanValue, loanStatus from loans`
2. `For each loan record selected`
   a. `Execute γ`
   b. `Accumulate the total`

- Time taken = $(0.8 + \beta)$ ms x 10^6

Push Algorithm Overview



Push Algorithm Overview



Gather Approver Details (γ)

```
1. select appExecCode
   from   approvals
   where  appLoanId = loanId
2. For each approval record selected
   a. Add loan data to corresponding
      reportRow
```

- Time taken = 1.5 ms + 0.8 ms = 2.3 ms

Select ReportRow Efficiently

- To achieve an efficient selection of the reportRow given an execCode
  a. Sort reportRow table and use a binary search
  b. Store the reportRows in a tree
  c. Use a hash table
- The sort and tree orders the report by execCode

Time Cost

Procedure	Time (hours)
Building the report structure (α)	1/3600
Gathering the information (β & γ)	0.86
TOTAL	0.86

Simplistic vs. PUSH

- Simplistic solution took about 1,120 hours!
- PUSH algorithm took about 0.86 hour
- For an overhead of as little as 52 kilobytes of memory, we get a performance enhancement of about 1,300 times!

Report Formatting

- Report title
- Date and time when the report was generated
- Pagination
- Column titles and alignment
- Totals line (when applicable)

Report Sample

Fri Jan 9 15:34:47 2004 MICROBANK LOAN ANALYSIS PAGE 1

EXEC-CODE	EXECUTIVE NAME	LOAN COUNT	LOAN VALUE	NLOAN COUNT	NLOAN VALUE
100000	PBJD SBXAM CYMCWT	210	90840.46	67	28631.97
100001	SEKK WZPR BRSRN	197	88837.56	54	27397.00
100002	ZHCICU RVHFZY PJFYU	212	89599.58	87	38042.98
100048	NDPVN WMZQ APIXIE	190	84504.00	58	26215.51
100049	PVCA OEFD YZNJNV	199	81652.14	64	27064.90
100050	IGGL YLPT BXNCW	192	84466.92	58	24227.54
100051	XAKSKW OLHCZ GCFFFE	182	81665.69	67	27478.40

Fri Jan 9 15:34:47 2004 MICROBANK LOAN ANALYSIS PAGE 20

EXEC-CODE	EXECUTIVE NAME	LOAN COUNT	LOAN VALUE	NLOAN COUNT	NLOAN VALUE
100982	PSRNPT UOJCDE JWUH	199	86969.33	59	23804.09
100983	WDUCIS CDBSNZ RJITGY	240	111238.10	68	32790.29
100984	WKYDC LUAM RJJX	186	82179.52	54	22445.76
100996	FNYDQ JGGHY TTDYM	195	83758.93	51	19072.49
100997	MBCPA YKAJOQ KYLC	183	80930.49	54	23788.67
100998	VONY WGAI OYEUYD	201	93634.99	67	31974.24
100999	NSJE KZILXA MVAO	204	91451.79	56	24274.06
TOTAL		1000000	45151285.25	30322	13688101.39

Summary

- Analysis of database access strategies and costs
- Deciding between simplistic and sophisticated
- The PUSH reporting algorithm
- Using a data table to assemble report lines
- Consideration of space vs. time when designing a solution