z/TPFDF V1.1

TPF Users Group Fall 2007
Title: z/TPFDF Education

Name: Kevin Jones
Venue: Ongoing Education

AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Agenda

- **Why Are We Here?**
- **z/TPFDF Introduction**
- **Database Concepts**
- **Application Programming Interface (API)**
- **Utilities**
- **Benefits**
- **Installation and Migration**
- **Myth Busting**
- **Service Data Objects (SDO)**
- **Questions and Answers**

Why Are We Here?

- **Introduce individuals and new customers to z/TPFDF**
  - z/TPFDF is required co-requisite for z/TPF
- **Review the benefits of z/TPFDF**
- **Demonstrate why z/TPFDF continues to be a viable tool for implementing new applications**

z/TPFDF Introduction

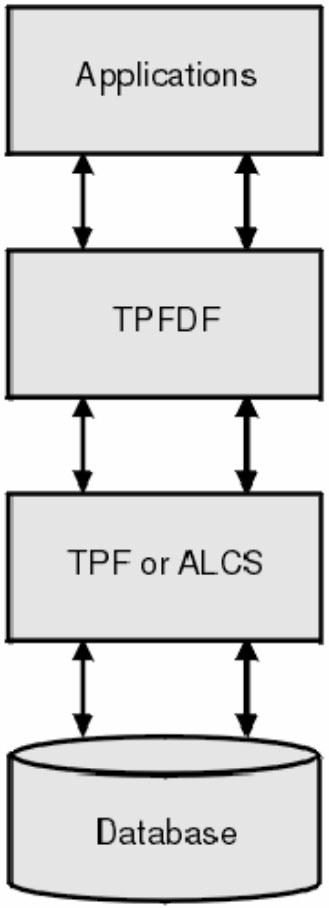
z/TPFDF is a database manager for <u>traditional</u> z/TPF applications that improves application programmer productivity with minimal performance impacts. It provides:

- A logical method of database organization
- Maps well to traditional TPF database layouts (hierarchical)
- A set of standardized assembler macros and C functions that form the application program interface (API)
- Central database routines for data access and manipulation
- Utilities for database maintenance and testing

Application programs are not sensitive to the physical implementation of the database

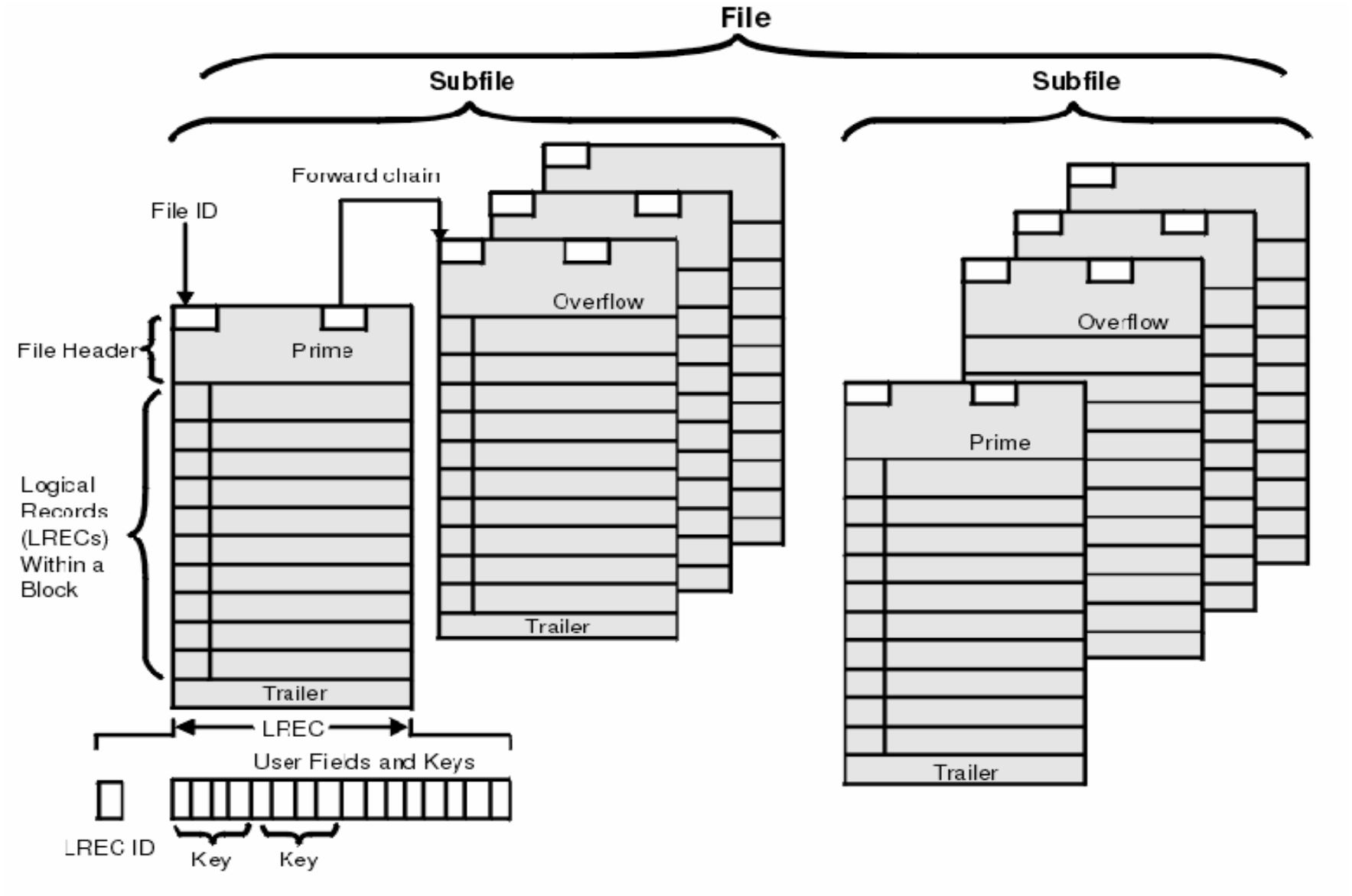
Introduction

- **Applications issue z/TPFDF APIs**
- **z/TPFDF issues traditional z/TPF find/file macros**
- **z/TPF reads and write the physical records in the database**



Database Concepts - Units of Data

- **Field – one unit of data (for example, a flight number)**
- **LREC (logical record) - smallest unit of data that an application can read, add or delete**
- **Block - collection of LRECs, along with a block header and optional block trailer (size is 381, 1055 or 4K)**
- **Subfile - prime block together with any attached overflow blocks**
- **File - collection of subfiles that share the same file ID (record ID)**
- **Database – a collection of files that share parent/child relationships in a hierarchical (“basic index”) structure**



Database Concepts - DSECT

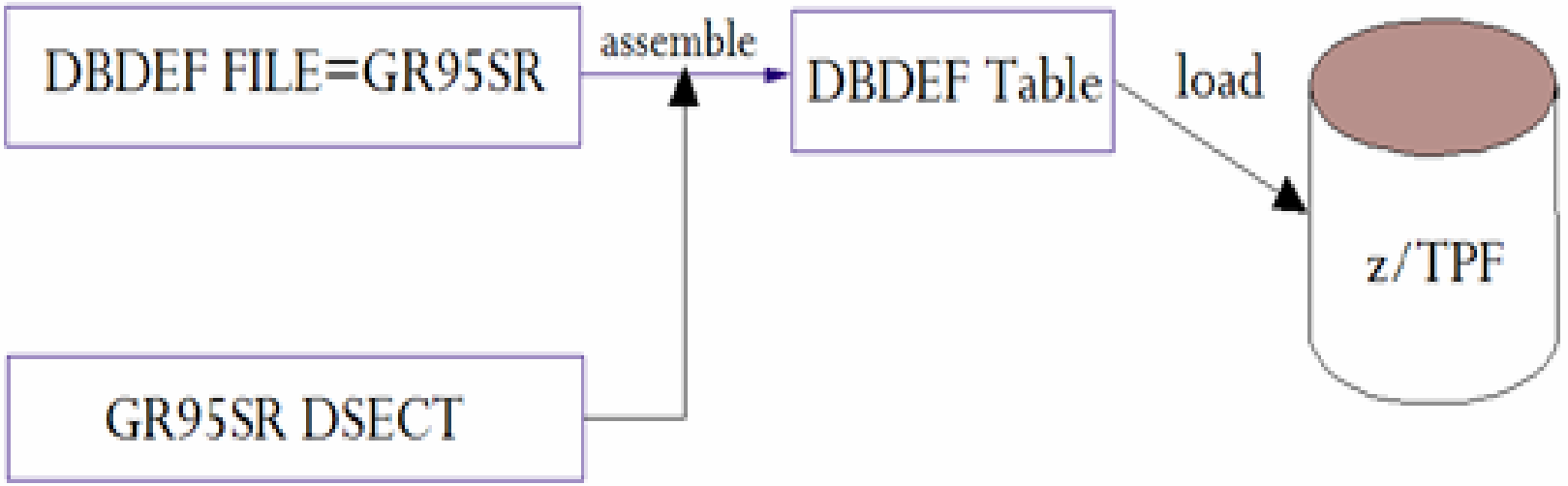
- **One DSECT must be coded for each file**
- **DSECTs include information such as the:**
  - File ID (record id)
  - Reference name
  - Block size
  - Logical Record IDs (also known as "primary keys")
  - Name, type and size of user fields
- **Samples provided with the z/TPFDF product**
- **One DSECT must be coded for each file**

```
&SW00WID SETC   'FD05'          FILE ID
&SW00WRS SETC   'L4'            BLOCK SIZE
&SW00ARS SETC   'L4'            ALTERNATE BLOCK SIZE
&SW02FIL SETC   'GR95SR'        FILE DSECT NAME
&SW00RBV SETC   '#TPFDBFF'      FILE ALGORITHM
&SW00OP1 SETC   '00000000'      OPTION BYTE 1
&SW00OP2 SETC   '00000110'      OPTION BYTE 2
&SW00OP3 SETC   '00000000'      OPTION BYTE 3
&SW00TQK SETC   '15'            HIGHEST TLREC

#GR95K80 EQU    X'80'           LOGICAL RECORD ID X'80'
#GR95K90 EQU    X'90'           LOGICAL RECORD ID X'90'
#GR95L80 EQU    GR95E80&CG1-GR95REC&CG1  LENGTH OF LOGICAL RECORD X'80'
#GR95L90 EQU    GR95E90&CG1-GR95REC&CG1  LENGTH OF LOGICAL RECORD X'90'

******** DESCRIPTION OF FIRST LOGICAL RECORD ID
GR95SEX&CG1 DS   CL1            MALE OR FEMALE
GR95SP1&CG1 DS   CL1            SPARE
GR95AGE&CG1 DS   CL2            AGE IN YEARS
GR95SP2&CG1 DS   CL1            SPACE
GR95NAM&CG1 DS   CL20           NAME AND INITIAL
GR95E80&CG1 EQU *               END OF LOGICAL RECORD WITH ID X'80'

******** DESCRIPTION OF NEXT LOGICAL RECORD ID
GR95SAL&CG1 DS   CL6            ANNUAL SALARY
GR95SP3&CG1 DS   CL1            SPARE
GR95JOB&CG1 DS   CL20           OCCUPATION
GR95E90&CG1 EQU *               END OF LOGICAL RECORD WITH KEY=X'90'
```

Database Concepts - DBDEF

- **DBDEF macro specifies the database definition of a file**
- **Parameters can override DSECT definitions**
- **Coded in realtime segment, assembled against the DSECT and loaded to z/TPF**
- **Consistent and central definition used by:**
  - z/TPFDF central database routines
  - Utilities

Database Concepts

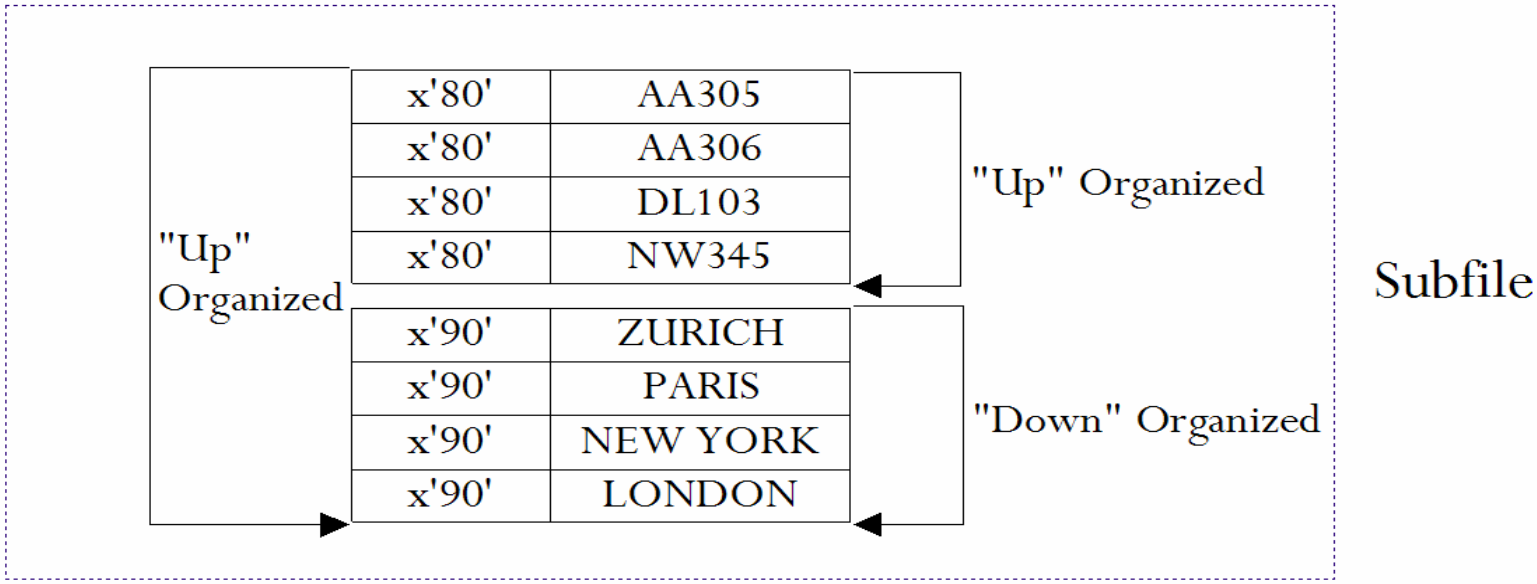


- ***Applications generally reference the DSECT***
- ***z/TPFDF references the DBDEF tables***

Database Concepts - Keys

- ***Default keys* are used to order LRECs in a subfile**
- **DBDEF can define up to 6 default keys**
  - always used to add LRECs
  - can also be used by other API macros and functions (for example, to read an LREC)

Sample Default Keys




```
DBDEF FILE=GR95SR,
      (PKY=#GR95K80,
      KEY1=(PKY=GR95K80,UP),
      KEY2=(R=GR95FLT,UP)),
      (PKY=#GR95K90,
      KEY1=(PKY=GR95K90,UP),
      KEY2=(R=GR95ORG,DOWN))
```

Database Concepts - Keys

- **Keys can also be explicitly coded on many API macros and functions (*selection keys*)**
- **Selection keys can be formed using:**
  - AND and OR Boolean connectors
  - Comparison operators ("greater than", "lesser than or equal", etc.)
- **Up to 180 selection keys can be specified**

Database Concepts - Keys

- ***Modification keys* can also be defined to modify fields in multiple LRECs with a single API call**
- **Can be combined with *selection keys***
  - for example, all employees living in a certain town can have their area code updated with a single API call

Database Concepts - Algorithms

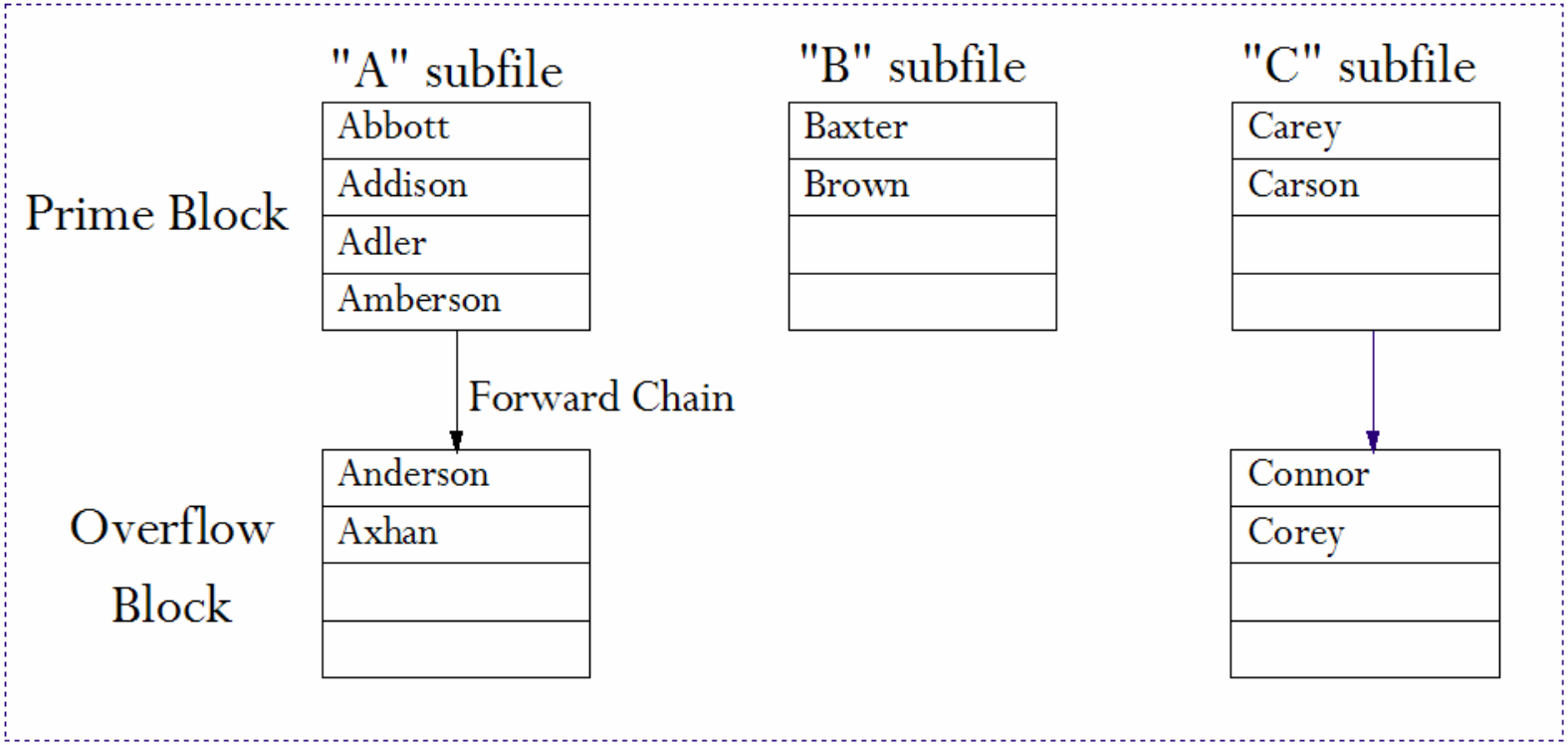
- **A rule that determines how a file is divided into subfiles**
- **Specified in the DSECT or DBDEF**
- **Target subfiles selected by specifying an algorithm parameter on an API macro or function**

DBRED REF=GR95SR,ALG=C'JONES'

- **Algorithms are generally based on alphanumeric strings, hashing values or user defined criteria**
- **Algorithms are also used in *basic indexing* – more on this later**

Database Concepts - Algorithm

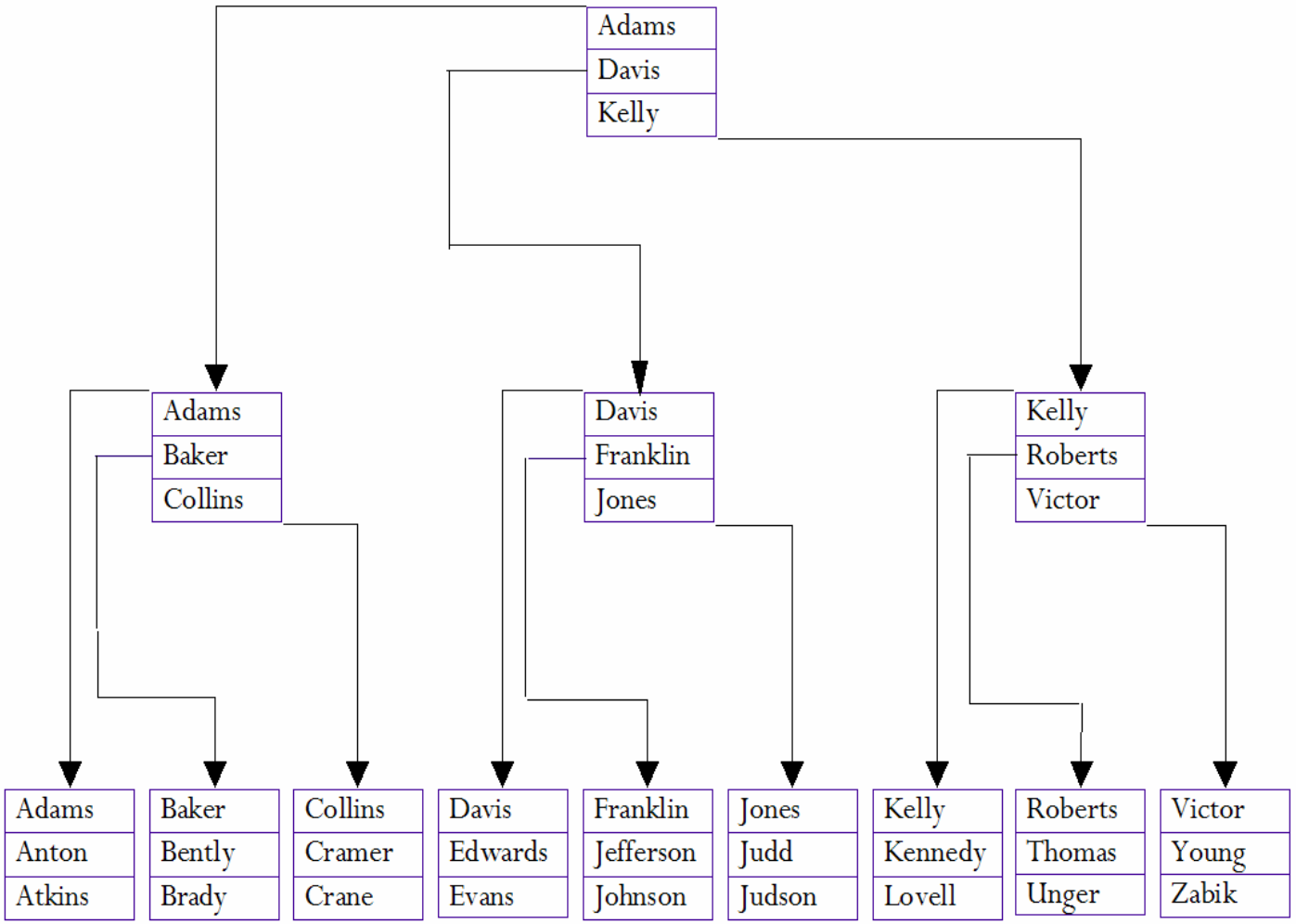
z/TPFDF File Using an Alphabetic Algorithm



Database Concepts - B+TREE Indexing

- **B+TREEs reduce the number of I/Os needed to access an LREC in overflow blocks**
- **Requires default keys in the DBDEF which are used to form the B+TREE index**
- **Files must be UP or DOWN organized**
- **Unlimited index size**
- **Index is transparent to application programs**

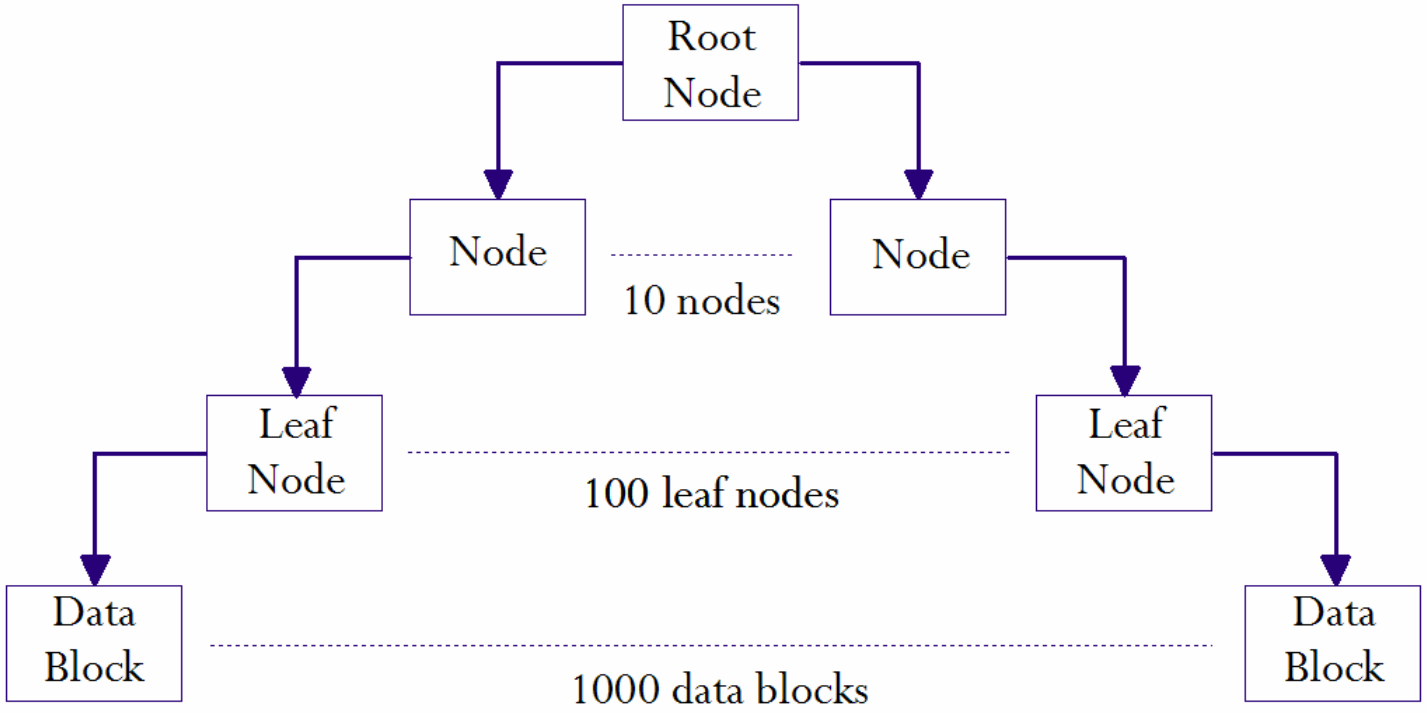
B+TREE index (root node and three leaf nodes



z/TPFDF subfile (prime block with 8 overflow blocks)

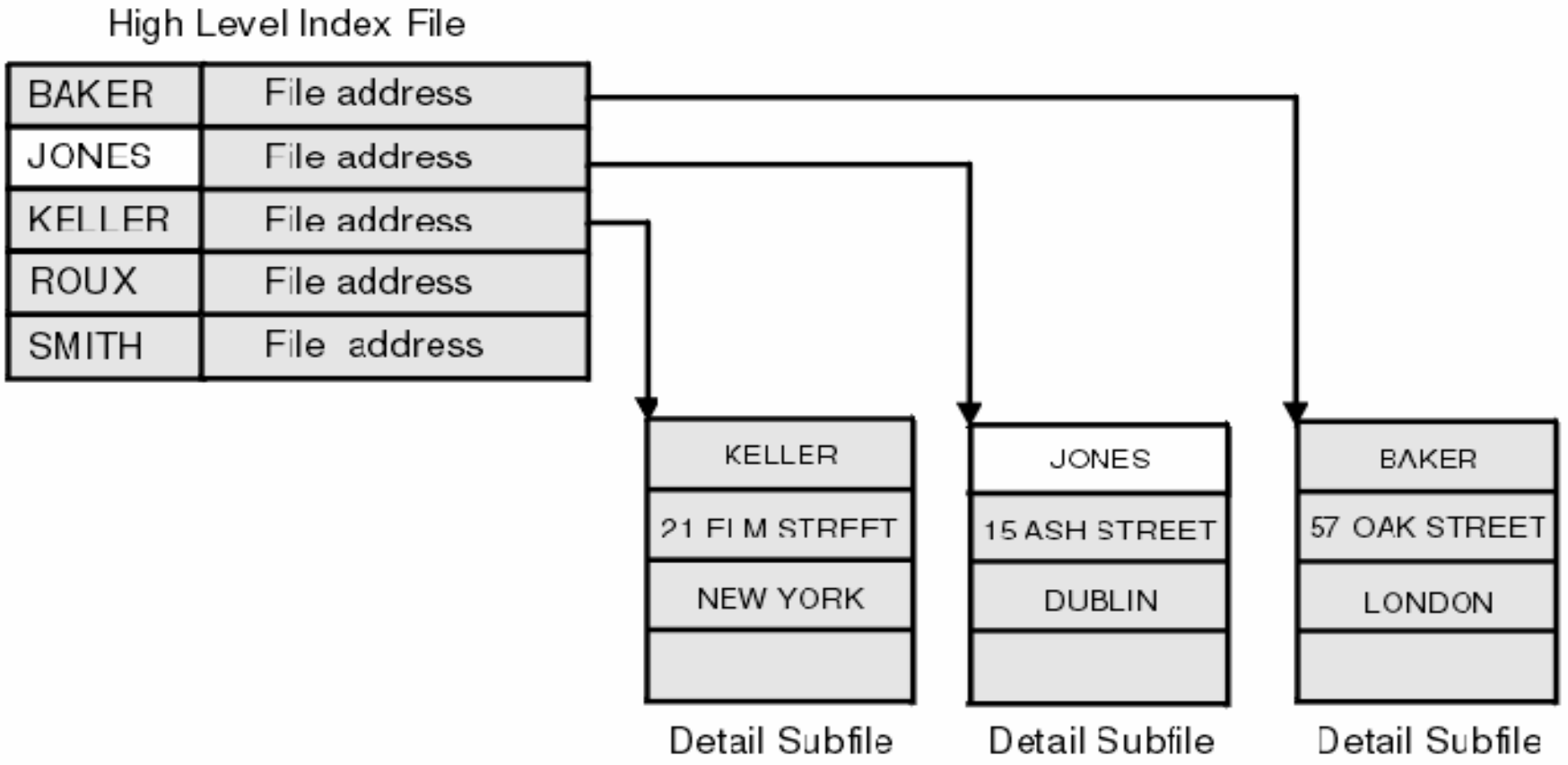
B+TREE I/O Savings

- **Assume the following database:**
  - 10,000 LRECs with 10 LRECs per block = 1000 data blocks
  - *Only 4 I/Os are required to locate any LREC*
  - *Only 5 I/Os would be required if database grew 10x*



Database Concepts - Basic Indexing

An alternate method of distributing LRECs across subfiles



Database Concepts - Basic Indexing

- **Index is transparent to application, except to specify the subfile**
  - DBOPN REF=GR95SR,ALG=C'JONES'
- **Any type of hierarchical structure is possible**
  - One high-level index with one detail file
  - Multiple indexes referring to one detail file
  - One high-level index referring to multiple detail files
  - Multiple levels of indexing

Application Programming Interface

- **The API allows a programmer to code applications without knowing the physical layout of the database**
- **Error checking is also performed by the z/TPFDF central database routines**
- **Macros (assembler) and functions (C language) are available to perform many different database operations**
- **The API generally works on LRECs within a subfile, but options are available to:**
  - access LRECs across subfiles
  - manipulate blocks, subfiles and files

Assembler Macro	C Function	Description
DBADD	dfadd	Add a Logical Record to a Subfile
DBADR	dfadr	Provide the File Address of a Prime Block
DBCKP	dfckp	Checkpoint an open file in main storage to DASD
DBCLR	dfclr	Allow ECB Exit with Open Files
DBCLS	dfcls	Close a Subfile
DBCPY	dfcpy	Copy a subfile
DBCRE	dfcre	Create an empty subfile
DBIFB	dfifb	Locate the z/TPFDF control information for a file
DBDIX	dfdix	Delete Index References to a Subfile
DBDEL	dfdel	Delete One or More Logical Records
DBDSP	dfdsp	Display Logical Records from a Subfile
DBFRL	dffrl	Ensure an ECB Data Level Is Free
DBIDX	dfidx	Create an Index Reference
DBKEY	dfkey	Activate a Keylist

Assembler Macro	C Function	Description
DBMRG	dfmrg	Merge Logical Records from Two Subfiles
DBMOD	dfmod	Indicate an LREC has been modified or modify LRECs that match previously established keys
DBOPN	dfopn	Open a subfile
n/a	dfopt	Set options in an open subfile
DBRED	dfred	Read a logical record
DBREP	dfrep	Replace a logical record with a new logical record
DBRET	dfret	Save a reference to the current logical record
DBRST	dfrst	Restore a Subfile
DBSRT	dfsrt	Sort a Subfile
DBSPA	dfspa	Allocate space for an opened file
DBTLD	dftld	Write a Subfile from Main Storage to DASD
DBTLG	dftlg	Write a Subfile to Tape
DBTRD	dftrd	Read a Subfile from an input tape
DBUKY	dfuky	Generate a Unique Key for Use in Logical Records

Utilities - File Maintenance

- **Many different functions, including:**
  - Display z/TPFDF tables
  - Display information about a file
  - Display contents of a subfile
  - Add or delete records from a subfile
- **Useful to make database changes without having to code an application**
- **File maintenance commands are ZUDFM**

Utilities - Data Collection

- **Separate from z/TPF Data Collection (ZMEAS)**
- **All data online - no offline component**
- **Provides data on:**
  - Accesses to z/TPFDF files
  - Number and frequency of each API call
- **Search criteria available to filter results and identify "problem" files**
- **Data Collection commands are ZUDFC**

Utilities - Capture/Restore (CRUISE)

- **Separate from z/TPF Capture/Restore**
- **Databases are captured and restored logically without regard to their physical layout**
- **Parameter tables define the data to be captured and restored**
- **CRUISE can also:**
  - verify database integrity
  - report database statistics
  - pack data blocks

Utilities - Recoup

- **Extension to z/TPF Recoup phase one**
- **Databases chainchased using DBDEFs**
  - Not necessary to code TPF descriptors
- **File addresses are included in subsequent recoup phases, same as any pool record found by z/TPF recoup phase one**
- **Additional ZRECP commands are available**

Benefits

- **The strength of z/TPFDF is that it complements traditional z/TPF find/file programming:**
  - significant application programmer productivity improvements
  - minimal performance impacts
  - database changes can be made more easily without updating every application
- **Initial database design is important**
  - allows subsequent stages of development and maintenance to proceed more effectively

Benefits - Productivity

- **Significant application logic common to z/TPF find/file is centralized in z/TPFDF**
  - Manage physical database layout
  - Manage database indexes
  - Error recovery
  - Sort and merge data
  - Use of commit scopes to ensure consistent database updates
  - Copy and restore data
  - Display data
  - Use keys to locate data

Benefits - Productivity

- **Customers report productivity improvements versus traditional z/TPF "find/file" applications:**
  - design 30%
  - code 50%
  - test 50%
  - maintenance 80%

Benefits - Performance

- **Customer benchmarks of typical application activities showed no performance degradation with z/TPFDF**
  - z/TPFDF performance is directly related to database design
- **Production systems exist that issue an average of 1 million TPFDF macro calls per second at peak**

Actual Customer Data Collection at 90% of Peak

```
17SEP07  09.30.01-09.35.01  - AVERAGE PER SECOND  - TOTAL

DBADD: 77847.6 DBADR:     0.8 DBCKP:   531.6 DBCLS: 35157.9
DBCPY:    44.9 DBCRE:   821.6 DBDEL:  7759.3 DBDIX:     0.2
DBDSP:      -- DBIDX:    13.6 DBKEY: 26958.4 DBMOD: 37924.1
DBMRG:     0.0 DBOPN: 35157.9 DBRED:650739.9 DBREP:  4696.1
DBRET: 24846.3 DBRST:    25.5 DBSPA:  5984.3 DBSRT:    39.0
DBTLD:      -- DBTLG:    55.2 DBTRD:      -- DBUKY:   424.8
DDA   :     --

FILNC:  1762.4 FILEC:     0.2 CFILE:   988.9 PFILE:  4140.8
TWRTC:    97.0 GETFC:  1948.4 RELFC:  1161.8 DETAC: 68184.7
ATTAC: 69271.8 GETCC:  8297.4 RELCC: 41260.1 CFIND:  9042.9
PFIND: 33218.5
```

Installation and Migration

- **z/TPFDF is a separate product from z/TPF**
- **z/TPFDF is a required co-requisite for z/TPF**
  - allows IBM to more easily port customer code written using z/TPFDF
  - allows IBM to use z/TPFDF without the need for conditional coding
- **PUTs are on the same schedule and numbering as z/TPF**
- **z/TPFDF PUTs are delivered in the same format as z/TPF PUTs**
  - installed into a "TPFDF" subdirectory in the z/TPF hierarchy

Installation and Migration

- **z/TPFDF PUTs typically have co-requisite and prerequisite APARs with z/TPF**
- **Customers not using TPFDF 1.1.3 have some additional migration steps when installing z/TPF and z/TPFDF:**
  - Largely RIAT and FACE table changes, along with a few initialization commands
  - Procedures are listed in the z/TPF migration documents
- **Once installed and initialized, z/TPF use of z/TPFDF databases will be transparent to new z/TPFDF customers**

Installation and Migration

- **New z/TPFDF customers looking to use the new functionality should consider:**
  - z/TPFDF and traditional databases can coexist:
    - Databases can reference each other
    - Applications can issue z/TPFDF and traditional macros
    - In certain cases, z/TPFDF can directly manipulate traditional files

Installation and Migration

- Database Administrators play a crucial role:
  - Responsible for overall database design
  - Codes or inspects DSECTs and DBDEFs
  - Monitors database performance and usage, making necessary adjustments
- z/TPFDF education is offered by third-party vendors

Myth Busting

- **z/TPFDF is not just for airlines**
  - TPFDF 1.1.3 is used by approximately 75% of all TPF 4.1 customers worldwide
  - Customers from across all industries are represented
- **z/TPFDF continues to be enhanced**
  - Large Logical Records
  - FARF6
  - z/TPF compatibility and leveraging
  - Service Data Objects (SDO)

Myth Busting

- **Tooling is being deployed to help debug TPFDF applications within the TPF Toolkit**
  - Logical record displays
  - Display of SW00SR information
- **Customers are deploying new applications written in TPFDF**
- **Data will not be forever trapped in a proprietary database manager accessible from z/TPF only**
  - Data will accessible through Service Data Objects (SDO)

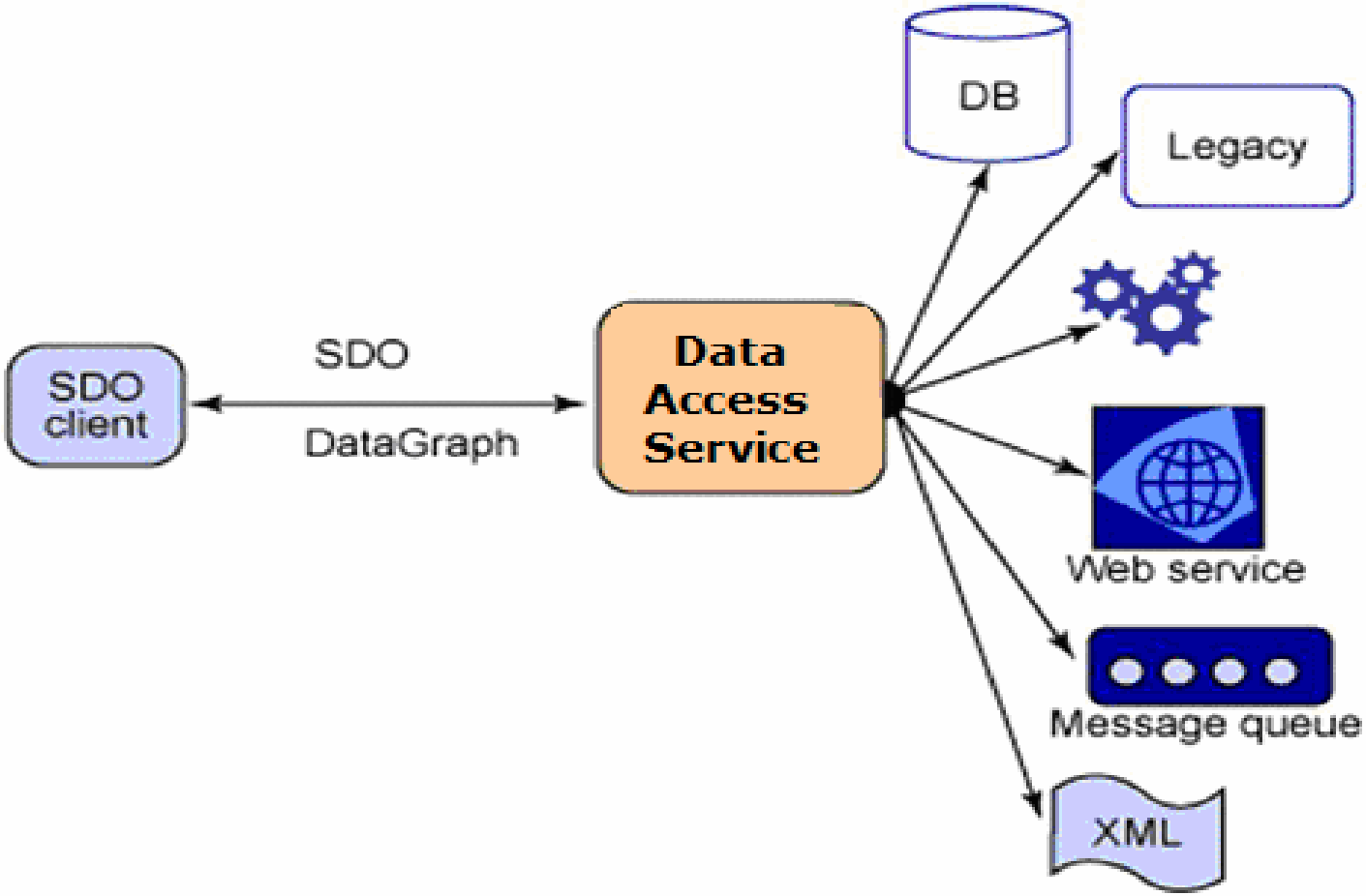
Service Data Objects (SDO)

- **Service Data Objects (SDO) is a new model of data access**
  - Complementary technology to Service Oriented Architecture (SOA)
- **Convenient and generic way to access data**
  - Common unifying format for exchanging data between services
  - Not tied to the data organization, like SQL to relational databases
  - Object-oriented, thus maintenance is easier

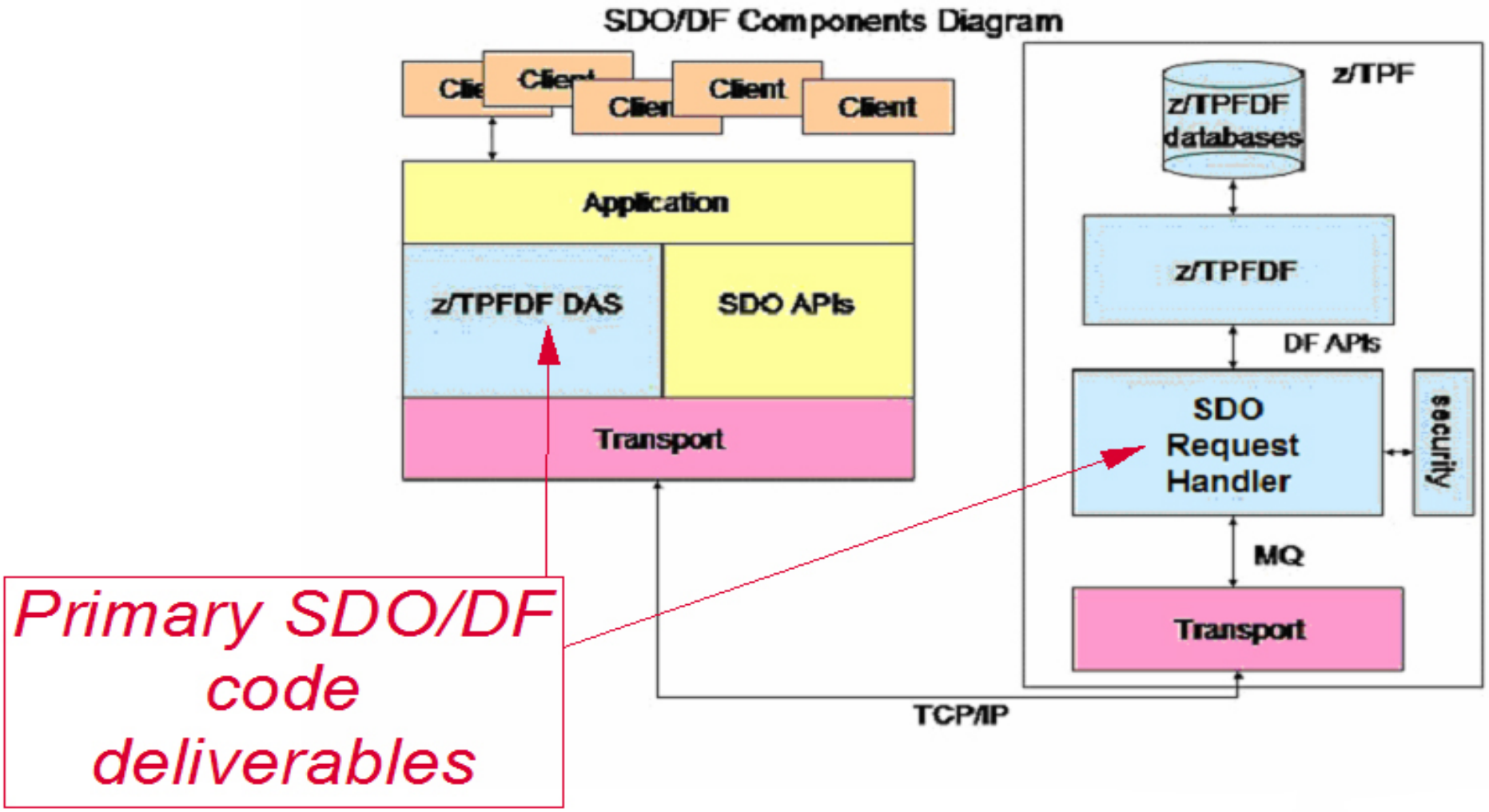
Service Data Objects (SDO)

- **SDO access to z/TPFDF will provide remote client applications access to z/TPFDF databases**
- **IBM is planning to provide a series of Java APIs for remote client applications**
  - These APIs will form a Data Access Service (DAS) for z/TPFDF databases
- **Customer applications will then be able to use the z/TPFDF DAS APIs together with SDO APIs to access and update data in z/TPFDF databases**

SDO in SOA



SDO and z/TPFDF



Questions and Answers